

“十五”国家重点电子出版物规划项目·计算机知识普及和软件开发系列



**编程宝典**  
**2002**  
**6**

北京希望电子出版社 总策划  
张龙卿 编写

# Delphi 6.0

## 数据库 深入编程技术

作者精心设计 14 个完整范例深入描述  
Delphi 6 开发数据库的方法和技巧，  
重点突出，实用性强

 **中国科学出版集团**  
 **北京希望电子出版社**

“十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列

编程宝典 2002 (6)

# Delphi 6 深入编程技术

北京希望电子出版社 总策划

张龙卿 编写

## 本版特点

作者精心设计 14 个完整范例深入描述

Delphi 6 开发数据库的方法和技巧，

重点突出，实用性强

## 适合对象

- 面向初中级读者
- 对高级用户也有重要参考价值



**北京希望电子出版社**

Beijing Hope Electronic Press

**[www.bhp.com.cn](http://www.bhp.com.cn)**

2002

## 内 容 简 介

这是一本面向初、中级读者的自学指导书，作者通过精心设计近 30 个（完整范例 14 个）范例深入介绍 Delphi 6 开发数据库的方法和技巧。全书由 14 章构成，主要内容包括 Delphi 6 基础；开发数据库应用程序，开发 BDE 数据库应用程序；开发 ADO 数据库应用程序；开发 InterBase 数据库应用程序；dbExpress 开发跨平台数据库程序；使用 Database Desktop 和 Datapump；数据库综合实例；打印报表；Decision Cube 面板与数据分析；界面设计常用功能；处理应用程序的异常；建立跨平台应用程序；跨平台应用程序实例，

本书内容新、范例丰富、重点突出、实用性强，不但是从事用 Delphi 6 进行数据库开发的广大初、中级编程员的自学指导书，对高级程序设计人员也有重要的参考价值。

本 CD 为配套书。

系 列 盘 书： “十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列  
编程宝典 2002（6）

盘 书 名： Delphi 6 深入编程技术

总 策 划： 北京希望电子出版社

文 本 著 者： 张龙卿

C D 制 作 者： 希望多媒体开发中心

C D 测 试 者： 希望多媒体测试部

责 任 编 辑： 但明天

出版、发行者： 北京希望电子出版社

地 址： 北京中关村大街 26 号，100080

网址: [www.bhp.com.cn](http://www.bhp.com.cn) E-mail: [lwm@hope.com.cn](mailto:lwm@hope.com.cn)

电话: 010-62562329, 62541992, 62637101, 62637102, 62633308, 62633309

（图书发行和技术支持）

010-62613322-215（门市） 010-62547735（编辑部）

经 销： 各地新华书店、软件连锁店

排 版： 希望图书输出中心 全卫

C D 生 产 者： 北京中新联光盘有限责任公司

文 本 印 刷 者： 北京双青印刷厂

开 本 / 规 格： 787 毫米×1092 毫米 1/16 开本 20.5 印张 470 千字

版 次 / 印 次： 2002 年 1 月第 1 版 2002 年 1 月第 1 次印刷

本 版 号： ISBN 7-900088-08-3

印 数： 1-5000

定 价： 35.00 元（本版 CD）

说明：凡我社产品如有残缺，可执相关凭证与本社调换。

# 前 言

Delphi 6 可以直接开发同时运行在 Windows 和 Linux 平台上的应用程序。前段时间 Borland 公司（即 Inprise 公司）发布的 Kylix 应用程序，实际上是 Delphi6 的 Linux 版本，只能用于开发运行在 Linux 上的应用程序。对于习惯于使用 Windows 开发环境的许多开发人员来说，如果要在 Linux 下使用 Kylix 开发应用程序，可能还需要很长时间的熟悉过程。而使用 Delphi6 开发跨平台的应用程序却没有任何障碍，熟悉 Delphi 的用户仍可以像以前那样使用它在 Windows 下开发应用程序。现在，Delphi6 可以建立两种类型的应用程序：一种是使用 VCL 组件库开发的应用程序，它只能运行于 Windows 平台上，这主要是保证与以前版本的兼容；另一种是使用 CLX 库开发的应用程序，这种应用程序既可以运行在 Windows 平台上，也可以运行于 Linux 平台上。使用 CLX 与 VCL 开发应用程序的方法完全相同，它们具有的大部分组件的名称、属性、方法及事件等也非常类似，通过本书的学习，可以很快掌握它们的异同点，这样就可以迅速地使用 CLX 组件开发出跨平台的应用程序。这些应用程序如果要移植到 Linux 平台上运行，只需要使用 Kylix 重新编译即可，几乎不需要修改任何代码及界面。

本书首先讲解了使用 Delphi6 开发应用程序的基础知识，以便使初学者迅速入门。然后对各种数据库组件的属性、方法、事件，数据库应用程序的体系结构及开发不同数据库应用程序的方法进行了详细讲解，另外，还重点讲述了 CLX 及 VCL 库的异同点，同时说明了如何使用 CLX 库开发跨平台的应用程序。还讲解了如何进行数据图表分析、建立快速报表、处理异常及如何使用设计界面常用组件等内容。

本书的一个突出特点是，通过大量实例说明问题，同时每个实例都提供一些实用的技巧。对一些组件的使用方法及开发应用程序中遇到的许多难题，都进行了透彻的剖析，以便读者在学习过程中少走弯路，提高学习和工作效率。

如果要学习使用 Delphi6 开发数据库应用程序和跨平台应用程序，本书可能是最合适的选择，它不但可以作为入门教程，也可以作为参考书。

本书由银河科技文化公司编写，另外，北京的欧洋、张劲、刘凯平、黄木平和深圳的梅红威参与了部分章节的编写工作，并提供了许多参考资料，在此一并表示感谢。

编 者

## 目 录

- 第 1 章 Delphi 6 基础
- 第 2 章 开发数据库应用程序
- 第 3 章 开发 BDE 数据库应用程序
- 第 4 章 开发 ADO 数据库应用程序
- 第 5 章 开发 InterBase 数据库应用程序
- 第 6 章 dbExpress 开发跨平台数据库程序
- 第 7 章 使用 Database Desktop 和 Datapump
- 第 8 章 数据库综合实例
- 第 9 章 打印报表
- 第 10 章 Decision Cube 面板与数据分析
- 第 11 章 界面设计常用功能
- 第 12 章 处理应用程序的异常
- 第 13 章 建立跨平台应用程序
- 第 14 章 跨平台应用程序实例

## 声 明

本电子版不包括第 2 章内容，请参看配套图书相关章节。

# 第 1 章 Delphi 6 基础

大家知道，Delphi 在开发数据库及设计应用程序界面方面有着不同寻常的优势：开发简单、设计方便、容易上手、帮助完善。只要对编程略有基础，则使用 Delphi 开发一般的应用程序界面及数据库应用程序都易如反掌，所以它越来越受到程序员的青睐。业界人员都认为：执着的程序员使用 C++ 语言，聪明的程序员使用 Delphi。使用 Delphi 编程，往往可以使程序员的工作事半功倍。

Delphi 6 在 Delphi 5 基础上又进行了很大的改进，它几乎可以实现 C++ 能够实现的所有功能，即不但设计应用程序界面方便快捷，在开发底层的应用程序方面也可以与 C++ 语言媲美。为了使大家迅速掌握 Delphi 6 的新增功能，下面首先介绍一下 Delphi 6 的新特点。

## 1.1 Delphi 6 的新特点

1. BizSnap 轻松创建基于 SOAP/XML 的 Web 服务，简化电子商务的集成。Delphi 6 无缝集成了基于 SOAP 的 Web 服务和 XML 数据交换技术，简化新一代电子商务的开发，是当前唯一在 Internet 上集成 Web 服务、B2B、B2C 和 P2P 的快速开发工具。

2. WebSnap 是基于组件的 Web 应用开发平台，支持 Apache、Netscape 和微软 IIS 等主流 Web 应用服务器。WebSnap 将 Delphi 应用及以当前流行的 HTML 开发环境——如 Dreamweaver、Frontpage、VBScript 和 JavaScript 所开发的网站无缝集成在一起。

3. DataSnap 创建高性能、提供 Web 服务组件，使客户端应用能够快捷地与 Internet 主流数据库连接。DataSnap 支持 Oracle、MS-SQL Server、Informix、IBM DB2、SyBase 和 InterBase 等主流数据库。它遵循业界标准的 SOAP/XML HTTP 协议，使客户端无需数据驱动和复杂配置，即可直接与其相连接。DataSnap 还支持 DCOM、CORBA、TCP/IP。

4. 构建单一代码的 Windows/Linux 应用。Delphi 6 与 Kylix 兼容。使用 Kylix，可在 Linux 平台上重新编译基于 Windows 平台的 CLX 应用；而利用 Delphi 6，即可在 Windows 上重新编译基于 CLX 组件的 Linux 应用。Delphi 6 包含 BaseCLX、VisualCLX、DataCLX 和 NetCLX 四个组件。

5. Delphi 6 与 AppServer 集成。Delphi 6 通过最新 SIDL 与 AppServer 连接。它为 AppServer 应用开发出高性能、具有丰富 GUI 环境的客户端应用，通过 Internet 将 AppServer 的 EJB 功能作为遵循业界标准的 SOAP/XML Web 服务发布到全球。

6. Borland VisiBroker for Delphi——支持客户端与服务器端的完整开发。Delphi 为 AppServer 和 VisiBroker 应用开发出高性能、具有丰富 GUI 环境的客户端和 Web 浏览器。在复杂的 IT 环境下，Delphi 6 所建立的 VisiBroker CORBA 对象能与任意 CORBA 客户端或对象进行互操作。

7. 支持最新 Windows 2000/XP 和 Office 2000。通过 ActionBands、ActionManagers 和 Shell Controls 创建的用户界面支持微软最新平台。使用户无需在繁琐的用户界面上耗费精力，便能轻松定制属于自己的 GUI 应用。

## 1.2 Delphi 6 的开发界面

打开 Delphi 应用程序，用鼠标单击系统菜单 File|New Application，则自动生成第一个项目文件，缺省文件名为 Project1.dpr，缺省表单为 Form1，代码编辑器中单元文件缺省名字为 Unit1.pas。这时的界面如图 1-1 所示。

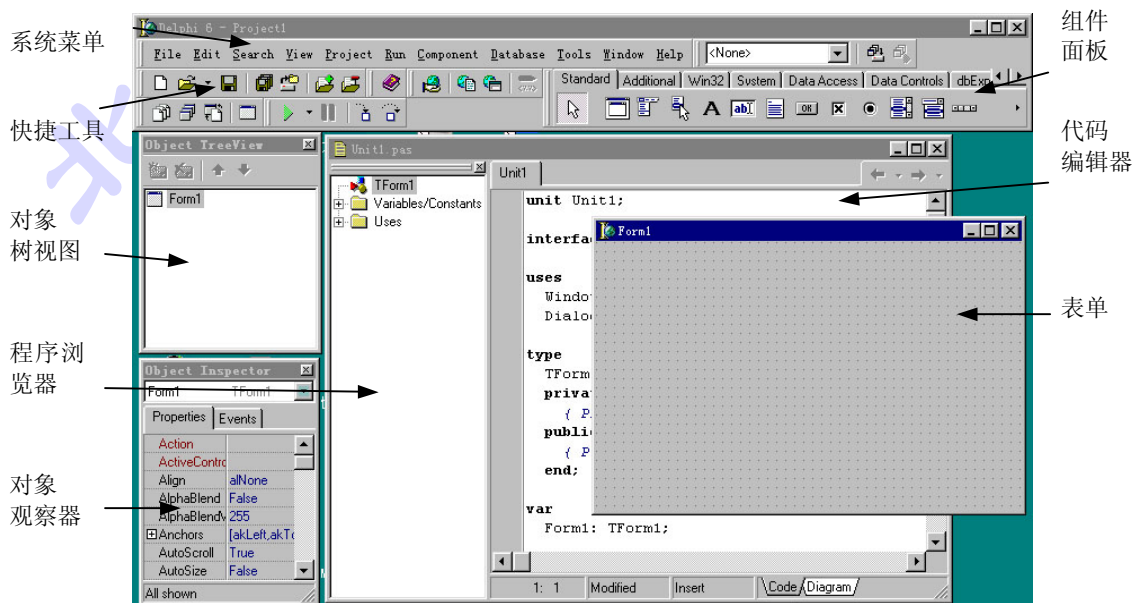


图 1-1 Delphi 6 开发界面

从图 1-1 中可以看出，Delphi 6 的集成开发环境（IDE）包括如下几部分：系统菜单（Menu）、快捷工具（Speed Bar）、组件面板（Component Palette）、对象树视图（Object TreeView）、程序浏览器（Code Explorer）、对象观察器（Object Inspector）、代码编辑器（Code Editor）和表单（Form），其中对象树视图是 Delphi 6 中新增加的功能。

下面分别说明各部分的作用。

1. 系统菜单（Menu） 系统菜单提供了 Delphi 6 集成开发环境中开发应用程序所需要的各种功能。

2. 快捷工具（Speed Bar） 快捷工具是部分系统菜单的另一种表现形式，通过这些工具可以快速打开一项菜单所提供的功能。其中，与 Internet 有关的三个快捷工具是新增加的，要显示或关闭一些快捷工具，可以通过在该面板上单击鼠标右键，此时会弹出一个菜单，如图 1-2 所示，然后根据需要选择或取消选择某个菜单，即可打开或关闭相应类别的快捷工具。

要区分某个菜单控制哪些快捷工具，可以反复打开再关闭每一个菜单，然后观察快捷工具面板的变化，即可知道其控制的快捷工具。

3. 组件面板（Component Palette） 使用 Delphi 开发应用程序时，大部分界面的设计

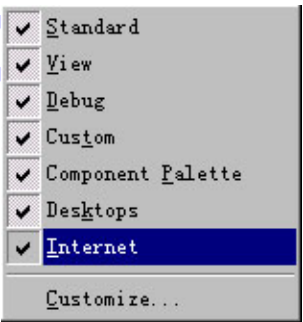


图 1-2 快捷工具弹出菜单



及功能的实现都是通过组件面板上的各个组件实现的。这些组件以类的形式出现，所以如果要使用某个组件，则相当于使用了该组件的实例。所有的组件根据其功能划分为多个类表，每一种类别的组件放置在特定的一个组件面板页上。Delphi 6 中定义了这些组件的两种标准格式：一种格式为可视化组件类型库 VCL (Visual Component Library)，这种格式的组件用来开发运行在 Windows 操作系统上的应用程序，Delphi 5 及以前的版本中的所有组件都属于该格式；另一种格式是跨平台的组件库 CLX (Borland Component Library for Cross Platform)，这是 Delphi 6 新增加的功能，用于建立可以同时运行在 Windows 和 Linux 操作系统上的应用程序。Linux 上对应 Delphi 6 的 Kylix 也使用该 CLX 组件库开发跨平台的应用程序。注意，该组件库是 Borland 产品中特有的功能。

由于使用 VCL 建立 Windows 应用程序与使用 CLX 建立跨平台的应用程序使用了不同的组件库，所以对应的组件面板也不同。

(1) VCL 组件面板 共包括 27 个组件面板，这些面板的作用如下：

- Standard (标准) 该组件面板包括一些 Windows 标准的控制项或选择菜单。
- Additional (附加) 该组件面板包括一些常用的控制项。
- Win32 该组件面板包括一些 Win 32 中常用的控制项。
- System (系统) 该组件面板包括一些与系统有关的控制项。
- Data Access (数据访问) 该组件面板包括一些存取数据库的不可见组件。
- Data Controls (数据控制) 该组件面板包括一些设计数据库应用程序界面使用的可见数据库组件。
- dbExpress 该组件面板包括一些建立跨平台数据库应用程序的组件。
- DataSnap 该组件面板包括一些用于建立多层数据库的组件。
- BDE 该组件面板包括一些使用 BDE (Borland 数据库引擎) 开发数据库应用程序时使用的数据库控制项。
- ADO 该组件面板包括一些使用 ADO 开发数据库应用程序时使用的数据库控制项。
- InterBase 该组件面板包括一些开发数据库应用程序时使用的数据库控制项。
- WebServices 该组件面板包括一些通过 SOAP 开发 Web 服务的组件。
- InternetExpress 该组件面板包括一些用于建立以 Web 浏览器为基础的瘦客户机应用程序组件。
- Internet 该组件面板包括一些协助开发和管理 Internet 应用程序的组件。
- WebSnap 该组件面板包括一些可以快速建立 Web 应用程序的组件。
- FastNet 该组件面板包括一些用于开发及管理网络应用程序的组件。
- Decision Cube (决策立体图表) 该组件面板包括一些分析和显示数据的组件。
- QReport (报表) 该组件面板包括一些建立打印报表所需要的各种组件。
- Dialogs (对话框) 该组件面板包括一些建立各种对话框的组件。
- Win 3.1 该组件面板包括一些兼容 Windows 较早版本的组件。
- Samples (例子) 该组件面板包括一些例子组件。
- ActiveX 该组件面板包括一些实现 ActiveX 功能的组件。
- COM+ 该组件面板包括一些用于开发兼容微软 COM+ 组件的控件。



- Indy Clients (Indv 客户) 该组件面板包括一些建立 Indv 客户的组件。
- Indy Servers (Indv 服务器) 该组件面板包括一些建立 Indv 服务器的组件。
- Indy Misc 该组件面板包括各种实现 Indy 服务的组件。
- Servers (服务器) 该组件面板包括一些与微软的 Office 套件相关的组件。

(2) CLX 组件面板 共包括 14 个组件面板, 其中大部分面板的名字与 VCL 组件面板的名字相同, 其包括的组件也实现类似的功能, 只有 Common Controls 组件面板是 CLX 特有的, 它包括与 VCL 中的 Win32 组件面板相似的组件, 也实现类似的功能。

4. 对象树视图 (Object TreeView) 对象树视图是 Delphi 6 中新增加的功能, 如图 1-3 所示, 它以树状图表显示放置在一个表单、数据模块或框架中的所有可见或不可见组件。对象树视图显示组件之间的逻辑关系, 比如同属关系、父子关系和属性联系。一个表单与放置在该表单上的按钮之间就是父子关系; 一个属性集与它的 FieldDefs 属性就有联系。有一些联系是隐含的, 比如数据集组件与它的属性。可以通过将一个组件拖拉到另一个组件的顶部来建立其他的关系, 只要它们之间有建立联系的可能性即可。

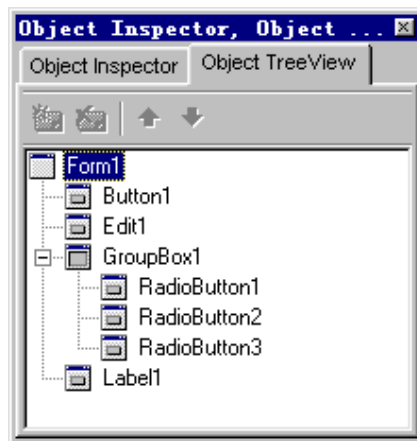


图 1-3 对象树视图

比如, 在表单中放置了一个面板 Panel 组件 Panel1 和一个标签 Label 组件 Label1, 则可以在对象树视图将 Label1 拖放到 Panel1 的下面, 这样 Panel1 和 Label1 之间就形成了一种父子关系, 同时也将表单 Label1 放置到了 Panel1 上面。这样在表单中拖拉 Panel1 时, Label1 也跟随移动。

在树状图中有一些节点以黑白颜色的图标显示, 则这些节点代表了隐含的组件, 例如, 一个 Dataset (数据集) 有一个缺省的 session (会话) 与它关联, 应用程序运行时会产生该会话。

如果一个节点使用黄色圆内的一个问号标注, 则表示该项还没有完全定义或存在问题, 比如, 一个没有设置 Dataset 属性的数据源就会显示该图标。

如果在一个项上双击鼠标, 则可以打开代码编辑器, 可以为事件处理器编写代码。

如果在对象树视图的一个项上单击鼠标右键, 则会显示一个上下文菜单, 该菜单是在表单、数据模块和框架中相同组件上面单击鼠标打开菜单的子集。

如果要在代码编辑器的 Diagram 页面建立一个图表, 则可以从 Object TreeView 中拖拉一些元素到 Diagram 页面中。

可以通过 View|Object TreeView 打开 Object TreeView 窗口。该窗口可以与对象观察器面板合并, 合并的方法是将对象树视图拖拉到对象观察器面板的中间位置, 如果在对象观察器的中间出现一个虚线框, 则应释放鼠标, 此时两个窗口就放置在一起, 如图 1-3 所示。当然, 该窗口也可以与代码编辑器等窗口放置到一起, 只要将其拖拉到合适的位置即可。

5. 代码浏览器 (Code Explorer) 代码管理器以树状的结构管理程序的代码, 它一般与代码编辑器放置在一起, 使用它可以很容易地在单元文件中进行导航。可以通过

View|Code Explorer 菜单打开该窗口，其显示如图 1-4 中右边窗口所示。

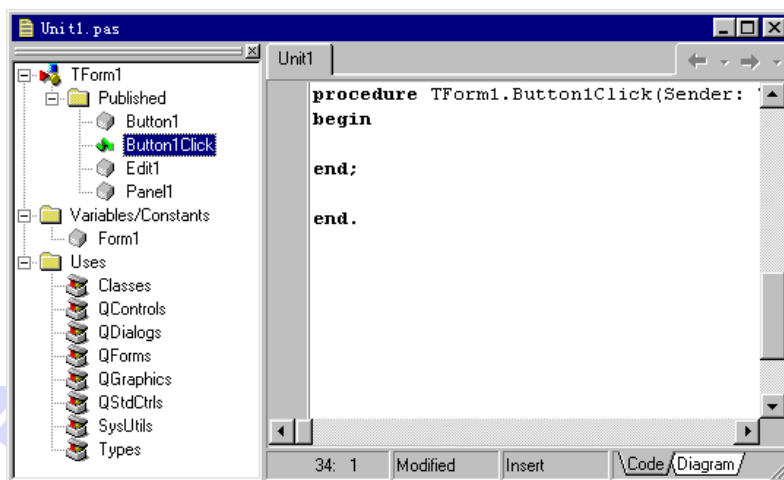


图 1-4 代码浏览器（右边）

代码浏览器窗口包含的树状图显示了在应用程序的一个单元文件中定义的所有类型、类、属性、方法、全局变量和全局程序，它也会显示在 uses 语句中包括的其他单元名称的列表，可以展开或收缩树的节点。

当选择一个新的单元文件时，代码浏览器中也会显示该单元文件中的内容。代码浏览器一次只能显示一个单元文件的内容。

如果在代码浏览器的一个项上面双击鼠标，则会在代码编辑器中导航到其对应的代码位置。

要调整代码浏览器的设置，可以选择菜单 Tools|Environment Options|Explorer。

6. 对象观察器 (Object Inspector) 它用来监视或设置表单中组件的属性或事件。可以按 F11 键或选择 View|Object Inspector 菜单打开它。当在表单、数据模块或框架中选择一个组件时，则对象观察器中会显示出该组件的属性。其显示如图 1-5 所示。

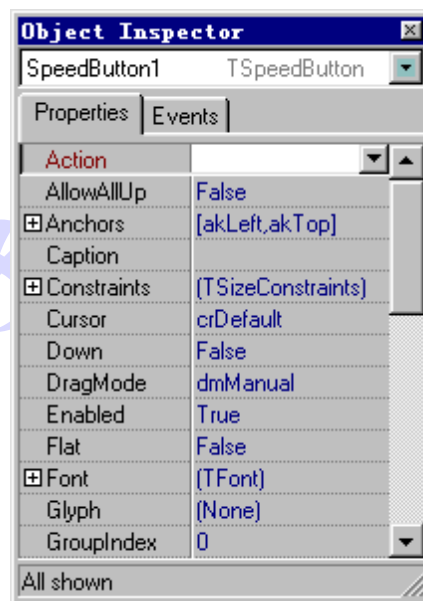


图 1-5 对象观察器

对象观察器包括三部分：对象选择器、属性标签页和事件标签页，它们的作用如下：

- 对象选择器 它位于面板的顶部，可以从下拉框中选择任何一个组件。
- 属性标签页 该标签页主要在应用程序的设计阶段设置各个组件的一些属性。属性用来描述组件的一些特征。Delphi 6 中对一些经常需要设置的或重要的属性使用褐色标注。
- 事件标签页 事件通常指一个对象的反应行为，例如，用户在按钮上单击鼠标，则会触发一个 OnClick 事件。如果要建立一个窗口，则会出发 OnCreate 事件。一

个组件执行的动作都是通过触发某个事件实现的，对应每个事件的代码相当于一个子程序。要为一个组件设置触发事件，只需要打开该组件的事件面板，在对应组件上双击鼠标，则会打开代码编辑器，并将光标放置在新加入的事件中，然后编写相应的实现代码即可。

Delphi 应用程序的建立步骤是，先在表单中添加组件，再设置组件的属性，并通过事件面板加入必要的事件，然后在代码编辑器中添加合适的执行代码。

7. 代码编辑器 (Code Editor) 顾名思义，代码编辑器用于编辑程序的代码，它具有比较完整的编辑功能，提供了多种编辑功能键。代码编辑器的显示如图 1-6 中右侧的窗口所示，它一般与代码浏览器放置在一起 (见图 1-6 左边窗口)。另外，在编译代码时如果遇到错误或警告等内容，则在代码编辑器的下面会显示出一个编译消息窗口 (见图 1-6 下面的窗口)，从中可以确定代码中引起错误的位置。

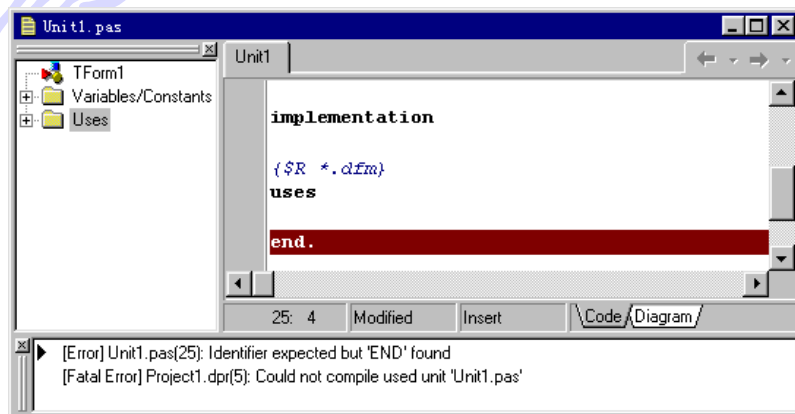


图 1-6 代码编辑器

在代码编辑器中单击鼠标右键，则会打开一个上下文菜单，该菜单提供了编辑代码需要的许多功能，同时还提供了一个 Properties 菜单用于设置代码编辑器的属性，比如可以使用它设置编辑器的显示方式、不同类型的代码使用的颜色等，通过这些属性的设置，可以使代码编辑器符合自己的编辑习惯。

另外，Delphi 提供了一种特别有用的功能，即如果要使用某些组件或对象的属性或函数，则应在代码中先输入该组件实例的名字，然后再输入一个小点，比如为 Form1.，此时它后面会显示出其提供的属性、函数或子对象的列表，同时不同类型的内容使用不同的颜色标注。这与 Delphi 5 是不同的。另外，可以按照子对象的作用范围或子对象名称的字母排列顺序对该列表框中对象进行排列。在列表中单击鼠标右键，并从弹出菜单中选择合适的菜单即可确定排列顺序。图 1-7 显示的是按照名称排列的列表。

通过子对象列表的功能，可以很容易地查找到当前对象的一些属性、函数及子对象等。

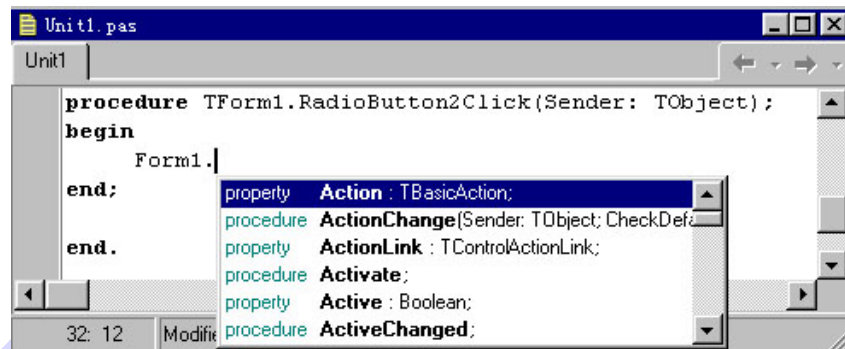


图 1-7 代码编辑器中显示的子对象列表框

Delphi 中的所有对象都是有归属关系的，父对象与子对象之间使用一个小点“.”表示其父子关系。比如，如果一个按钮 Button1 放置在表单 Form1 中，则在第二个表单 Form2 对应的单元文件 Unit2 中修改 Button1 的标题时，可以使用如下的代码：

```
Form1.Button1.Caption:="确定";
```

如果是在 Form1 的单元文件 Unit1 中修改 Button1 的标题，则可以省略 Form1。可见，一个小点“.”建立的联系，决定了一个对象的归属。

8. 对象图表 在图 1-1 中介绍设计界面的时候，没有提到对象图表，因为它是代码编辑器中的一个页面（Diagram）。这是 Delphi 6 新增加的功能，所以单独讲解该部分可能会更有利。其显示如图 1-8 所示，该图表示在一个组合框中放置了两个单选按钮，故它们构成了父子关系。

如果没有将组件从 Object TreeView 中拖放到 Diagram 页面，则它不会显示任何组件内容。通过拖拉，可以将多个组件和它们的子对象和属性放置到 Diagram 页面中。

- 如果选择多个组件直接拖放到 Diagram 页面中，则这些组件会垂直排列。
- 如果在按住 Shift 的同时拖拉多个组件到 Diagram 页面中，则这些组件会水平排列。
- 要重新排列 Diagram 页面中的组件，则需要重新从 Object TreeView 中拖拉它们。

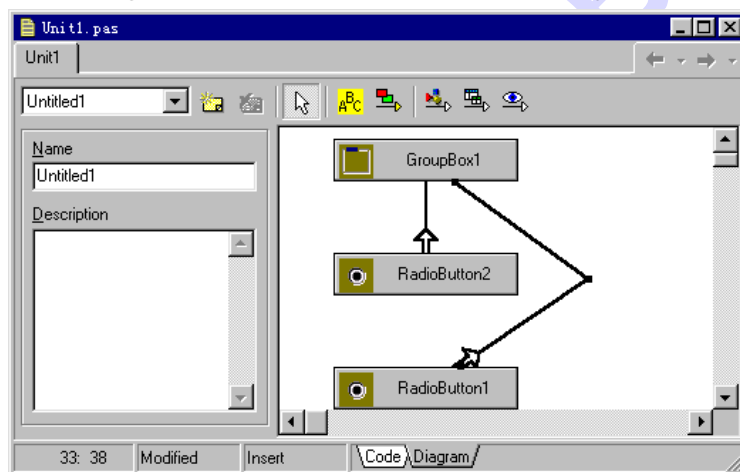


图 1-8 代码编辑器中的 Diagram 页面

Diagram 页面可以划分为三部分：上面一行提供了一些建立对象关联的工具及对图表

进行管理的工具；左下角部分是当前选择图表的名字及描述；右下角部分则显示了对象之间的联系。

(1) 工具图标的作用 Diagram 页面顶部一排快捷工具图标的作用如下：

-  该列表框中列出了已经建立的所有对象图表，可以从中选择一个进行编辑。
-  New Diagram (新建图表) 单击该图标，可以建立一个新的图表。
-  Delete Diagram (删除图表) 该图标用于删除一个图表。
-  Select Mode (选择模式) 单击该按钮，可以进入选择模式来选择组件。
-  Comment Block (注释块) 单击该图标会添加一个黄色的注释块，可以输入注释内容。
-  Allude Connector (隐含连接器) 选择该图标后，可以在 Diagram 页面上的对象之间连接隐含的连接关系，使用一个单向箭头将两个对象连接起来，对整个应用程序没有任何影响。所以一般使用该连接器连接注释块。
-  Properties Connector (属性连接器) 如果一个对象的属性值是另一个对象，则可以使用该图标建立两者的联系。比如将 DBGrid1 的 DataSource 属性设置为一个数据源对象 DataSource1，则将 DBGrid1 和 DataSource1 同时拖拉到 Diagram 页面时，会显示两者的关系。如果还没有建立关系，则可以通过 Properties Connector 建立连接。
-  Master/Detail Connector (主从连接器) 用于建立主从表之间的关联。
-  Lookup Connector (查找连接器) 建立查找字段与当前数据集的关联。

注意：Diagram 页面建立的关联也会作用到 Object TreeView 窗口中相应对象上面。

(2) 对象之间的联系 通过以上的各个快捷图标可以建立对象之间的联系，这些关联使用一些带箭头的线段表示。可以在一条线段的任何位置按下鼠标左键并拖拉鼠标，则可以使线段出现拐角（如图 1-8 所示）。

在保存单元文件时，会自动保存 Diagram 页面的内容，其文件扩展名是.DDP。如果要打印一个图表，则应先选择该图表，然后选择系统菜单 File|Print 即可。

9. 表单 (Form) 表单是放置组件的容器，所有的可见和不可见组件都可以放置到上面。表单也是显示给用户的最终界面，可以是窗口或对话框。应用程序运行时，用户与应用程序进行的交互，都是通过表单实现的。

表单本身也是一个组件，可以设置它的属性及触发事件。比如，当在应用程序显示表单时要触发一个事件，可以通过 OnShow 事件编写实现特定功能的代码。

与表单非常类似的是框架 (Frame) 和数据模块 (Data Module)，它们都是存放其他组件的容器，在应用程序中都不会显示。

框架用于保存一些重复使用的内容，这些内容一般是多个控件的组合。可以将整个框架作为一个组件保存到组件面板上，保存方法是在框架中单击鼠标右键，并从选择菜单中选择 Add to Palette，在弹出对话框中进行适当的设置，就可以将其保存到组件面板中重复使用。注意，当原框架中的组件进行了改变时，保存在面板上的组件也会做相应的改变。在表单中使用框架建立组件时，原框架中各个组件的触发事件仍然起作用，但是在插入到表单的实例中却无法看到这些触发事件。比如，在框架中插入了一幅图像、一个按钮，然



后将其保存到组件面板中。以后在表单中插入该框架生成的组件时，则会同时插入框架中的图像或按钮，也会使用它们具有的触发事件。

数据模块主要用于保存不可见的数据库组件，比如 DataSource、SQLDataSet、Table 等组件，以便集中管理和设置。

### 1.3 Delphi 应用程序的组成

Delphi 开发的应用程序或动态链接库是以项目（Project）的形式出现的，一个项目是多个文件的集合。其中一些文件是设计项目时生成的；另一些文件则是在编译项目时形成的。一般情况下，你不必对每个文件进行处理，尽管可以直接编辑大多数文件的源代码和脚本，但使用 Delphi 的集成开发环境及可视化工具更方便，也更可靠。

在打开 Delphi 时，会自动建立一个项目文件，如果使用 File|Save Project As 保存整个项目文件，则可以看到该文件包括如下文件：项目文件（.dpr）、项目选项文件（.dof，Kylix 中为.kof）、配置文件（.cfg，Kylix 中为.conf）、资源文件（.res）、表单文件（.dfm，跨平台应用程序中为.xfm）、单元文件（.pas）。其中前 4 个文件都使用项目的名字，比如为 project1；后两个文件使用单元的名字，比如为 Unit1。

1. 项目文件 每个项目都包括 Object Pascal 源代码，Delphi 将其编译为最终的应用程序或动态库。项目文件一般包含对使用的所有表单和单元的引用，在装载、保存或编译一个项目时，Delphi 查看项目文件就知道各类文件的作用。项目文件最终会生成.exe 或.dll 文件，前者是应用程序，后者是动态链接库。

**注意：**由于 Delphi 本身维护项目文件，所以一般不需要手工修改它。对项目文件的改变也可以通过 Project Manager（项目管理器）实现，可以由 View|Project Manager 打开它。

要查看项目文件的源代码，可以通过 Project|View Source 打开它，也可以由 View|Units 菜单或快捷按钮打开它。缺省生成的项目源文件代码如下：

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

该源代码各部分的作用如下：

- **program** 保留字 program 指出该项目是一个应用程序，其名字为 Project1；如果该项目是一个动态库，则会使用 library 保留字。
- **uses** 该语句告诉编译器当前项目中使用了哪些单元，并将其链接到项目上。在表单中添加新的组件时，与其有关的标准单元文件会自动添加到该语句中，所以一般不需要对其修改。如果要引用自己建立的单元，可以再使用一个 uses 语句，并将其放置在 {\$R \*.res} 上面，这样源代码显得比较清晰，这种情况一般出现在表单



单元文件或单独的单元文件中。

- **Unit1** 缺省生成的单元标识符，它对应缺省建立的表单 Form1，UNIT1.PAS 是包含该单元源代码的文件名，它与单元标识符必须一致，否则项目无法正确被编译。所以如果修改了单元标识符，则也应新的名字保存单元文件。
- **in** 该保留字告诉编译器如何找到每个单元的源文件，注释{Form1}表示该单元文件对应表单的名字。
- **{SR \*.res}** \$R 是一条编译器指令，它告诉编译器应该将与项目文件同名的资源文件\*.res (\*表示与当前的项目文件同名)链接到项目中。项目的资源文件包括项目的图标、图像等内容。
- **begin...end** 该部分是当前项目的主要源代码块，其中 Application.Initialize 语句用于对应用程序初始化；Application.CreateForm(TForm1, Form1)语句则用于建立参数中指定的表单，如果项目中有多个表单，则也会有多条与其对应的这种语句；Application.Run 语句会运行整个应用程序。

2. 单元文件 Delphi 的 Object Pascal 语言支持单独编译的代码模块，称为单元，即一个单独的源代码文件。使用单元可以提高代码在各项目之间的结构化和重用性。Delphi 项目中一般的单元文件都与一个表单对应，它包含了事件处理程序和表单的其他代码。但是单元文件也可以不与表单关联，即可以通过 File|New Unit 菜单建立单独存在的单元文件，这些单元文件可以被多个项目使用。例如，可以编写实现特定功能的过程、函数、动态库和组件，然后将其保存为一个单独的单元文件，然后将这些单元文件拷贝到保存其他项目的目录中，并使用 uses 语句引用这些单元文件，即可重用它们。

单元文件被编译时，会生成一个二进制的编译文件，其扩展名是.dcu，这些文件用于项目快速的编译和链接。

一个单元文件可以划分为接口和实现两部分。其中接口部分以 interface（接口）保留字开始，到 implementation（实现）关键字为止，该部分用于单元文件中使用的类或变量的声明等；implementation 关键字以后的代码是该单元文件中实现的功能，包括对各个组件事件的处理及实现不同功能的过程和函数等。

缺省生成的表单单元文件的源代码如下，后面都添加了注释：

```
unit Unit1;           //单元文件的名字
interface           //接口部分的开始
uses                //引用的标准单元文件
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;
type                //Delphi自动添加的类型声明，这是表单特有的
  TForm1 = class(TForm)
  private
    { Private declarations }    //该单元私有变量的声明
  public
    { Public declarations }      //声明公用类型
  end;                          //结束类型声明
var                  //声明变量或类的实例
  Form1: TForm1;
implementation       //程序代码实现功能部分的开始
```

```
{SR *.dfm}           //通过编译指令SR链接表单文件
end.                  //实现部分结束
```

3. 表单文件 表单文件一般以二进制的形式存储, Delphi 中其文件扩展名是.dfm, 在 Kylix 中其文件扩展名是.kfm。

可以在代码编辑器中打开一个表单文件并修改这些数据的文本版本, 将表单文件转换为文本形式的方法是在表单上面单击鼠标右键, 然后从打开的上下文菜单中选择 View as Text 选项。如果要将文本形式表单重新转换到以前的形式, 则可以选择 View as Form 选项。

Delphi 的每一个表单都有一个与其关联的单元文件, 单元文件中包括了处理表单上各个组件相应事件的代码。

4. 资源文件 Delphi 使用标准的 Windows 格式的资源文件, 它包括项目使用的图标等内容。缺省情况下, Delphi 的每个项目文件都有一个资源文件, 其扩展名为.res。

5. 项目选项文件 项目选项文件也与项目文件同名, 但使用的扩展名为.dof (Kylix 使用的扩展名为.conf), 它保存了通过 Project|Options 菜单打开的对话框中设置的各个选项。该文件是一个文本文件, 可以使用一个文本编辑器打开并编辑它, 但建议一般不要这样做。

6. 项目配置文件 项目配置文件保存着项目的配置信息, 它也使用与项目相同的名字, 但是其扩展名为.cfg。该文件也是一个文本文件, 可使用一个文本编辑器打开它, 但是一般不要对其修改。

7. 其他文件 当然, 一个复杂的项目可能还包括其他文件, 如果项目中使用了包 (package), 则需要将这些包的代码编译进运行时的包中。包源文件的扩展名是.dpk, 包的扩展名是.dpl。包的最主要作用是提高代码的重用性及应用程序的执行效率。当需要重新编译含有包的项目时, 包也在必要时隐式地进行重编译。

## 1.4 一个应用程序实例

要真正掌握 Delphi 开发应用程序的方法和步骤, 可以从最简单的实例开始。下面我们将建立并运行一个简单的实例。

### 1.4.1 建立项目文件

选择 File|New|Application 菜单, 建立一个新的应用程序。如果要建立跨平台的应用程序 (既可以运行在 Windows 上, 也可以运行在 Linux 上), 则应选择 File|New |CLX Application。

### 1.4.2 添加组件

1. 添加标签 在 Standard 组件面板上找到标记 A 的标签组件, 然后在 Form1 左上角单击鼠标; 或直接在该组件上面双击鼠标, 则一个标有 Label1 的标签出现在 Form1 表单上, 标签周围的黑点表示当前选中了组件。

2. 添加编辑框 在 Standard 组件面板上找到标记 ab 的编辑框组件, 然后在 Form1 中 Label1 右边单击鼠标; 或直接在该组件上面双击鼠标, 则编辑框 Edit1 会添加到 Form1 中。

3. 添加按钮 在 Standard 组件面板上找到标记 OK 的按钮组件, 然后在 Label1 下面插入一个按钮 Button1; 按照同样的方法, 再继续插入第二个按钮 Button2。

按住鼠标左键拖拉各个组件可调整其位置，用鼠标拖拉它们的边框可适当调整其大小。如果要同时调整多个组件的位置，可按下鼠标左键拖拉鼠标，此时会出现一个虚线框，当释放鼠标时，虚线框范围内的组件都会被选择，然后在虚线框内按下鼠标左键拖拉鼠标。

以上在 Form1 中添加了四个组件并调整完毕后的显示，如图 1-9 所示。

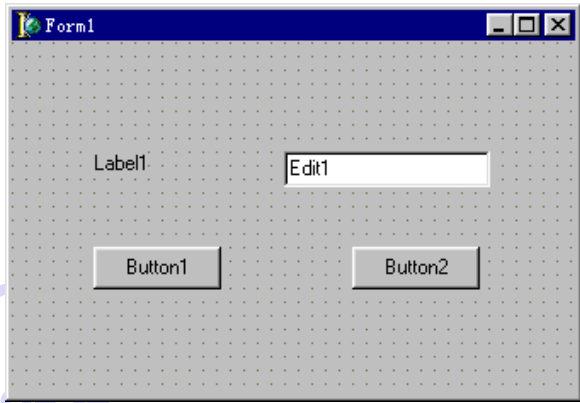


图 1-9 添加四个组件后表单 Form1 的显示

1.4.3 修改属性

在表单 Form1 上的空白位置单击鼠标，则会选中它，然后用鼠标单击系统菜单 View|Object Inspector 或按 F11 键打开对象观察器，再选择 Properties 标签页，然后就可以修改 Form1 的属性了。

其他组件设置属性的方法与之类似。设置 Form1 及它上面的 4 个组件的属性值见表 1-1。其中，“字体”属性修改的是字体类型、字体样式、字体大小三部分，分别用“/”隔开，以后章节将使用同样的方法设置字体属性，就不再提示。

注意：大部分组件都只需要修改几个关键属性，其他的属性使用缺省值即可。

另外，要调整组件的位置，可直接拖拉该组件；要改变组件的大小，可按下鼠标左键拖拉其边框。但是这些方法只能粗略调整组件的位置及大小。要进行精确调整，则需要设置组件相应的属性值。

表 1-1 设置组件的属性

组件名称	属 性	设 置 值	说 明
Form1	Caption	测试程序	标题
Label	Caption	请输入内容:	标题
	Color	clRed	颜色
	Name	L hello	名字
	Font	粗体/18 号/黄色	字体
Edit1	Text		编辑框显示文本

(续表)

组件名称	属 性	设 置 值	说 明
Button1	Caption	显示输入内容	标题
	Name	Btn_show	名字
	Font	楷体/粗体/小四	字体
	Height	33	高度
	Width	129	宽度
Button2	Caption	关闭窗口	标题
	Name	Btn_exit	名字
	Font	楷体/粗体/小四	字体
	Height	33	高度
	Width	129	宽度

要对齐多个组件或设置组件的大小一致,则先选择这些组件,然后通过 Edit|Align 或 Edit|Size 菜单打开的面板进行调整即可。比如调整 Form1 中两个按钮顶部对齐的设置对话框如图 1-10 所示。

以上所有组件的属性设置完毕后,表单的显示如图 1-11 所示。

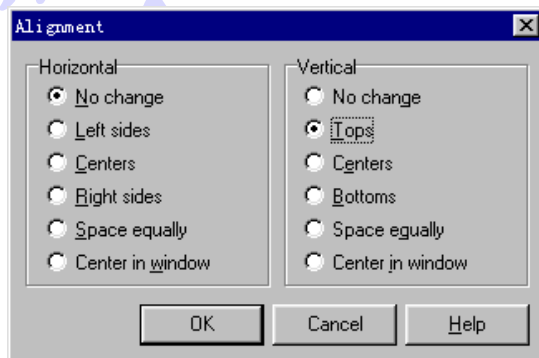


图 1-10 修改属性后的界面



图 1-11 修改属性后表单的显示

#### 1.4.4 添加代码

1. 设置按钮“显示输入内容”触发事件 我们希望在应用程序运行时,单击该按钮后,会弹出一个对话框显示编辑框中输入的信息,则可以通过 ShowMessage 函数显示该信息。

选中“显示输入内容”按钮，用鼠标单击“对象观察器”中的 Events 面板，在标有 OnClick 一行的右边双击鼠标，则会打开代码编辑器，此时光标正好放在 Procedure TForm1.Btn\_showClick(Sender: TObject)的 begin...end 之间，即该按钮触发事件的代码部分。直接在 Form1 中该按钮上双击鼠标，同样可激活该事件并进入代码编辑器。

现在给程序添加执行代码，先在光标指示位置输入几个空格，然后输入代码：ShowMessage，此时会显示出用黄色表示的该函数参数的提示信息，如图 1-12 所示，表示该函数只有一个字符串类型的参数。接着输入要显示的内容，完整的语句如下：

```
ShowMessage('您输入的内容是：'+Edit1.Text);
```

上面语句表示对话框显示的内容由两部分字符串组成：一部分是“您输入的内容是：”；另一部分是 Edit1.Text，即利用了编辑框 Edit1 的 Text 属性值，这也是一个字符串。两部分字符串通过字符串连接符“+”连接到一起。

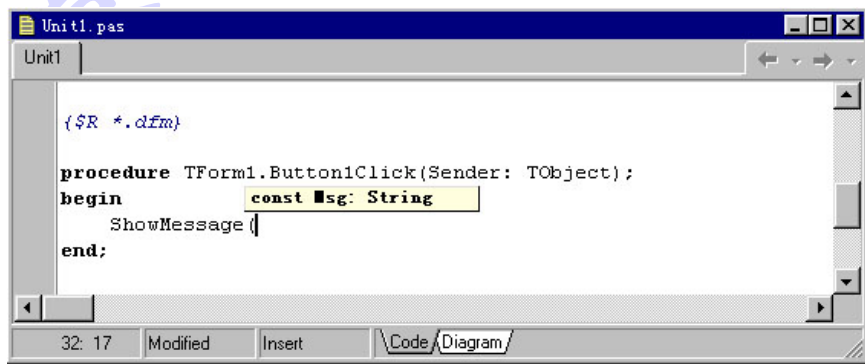


图 1-12 函数参数的提示信息

2. 设置“关闭窗口”按钮的触发事件 为了使程序在运行时，用户单击“关闭窗口”按钮后能退出整个程序，可以按照如下方法添加代码。

选中“关闭窗口”按钮，按照上面的方法为该按钮添加一个 OnClick 事件，然后在光标指示位置给该事件添加执行代码。先在光标指示位置输入几个空格，然后输入如下代码：Application.t，稍微等待一会儿，会弹出一个窗口，显示出以 T 开头的所有过程或属性等内容，如图 1-13 所示。现在可以使用向下的箭头键选择 Procedure Terminate，并按 Enter 键；或在该过程上面双击鼠标，则 Terminate 会自动添加到 Application.的后面，最后输入语句结束符“;”即可。

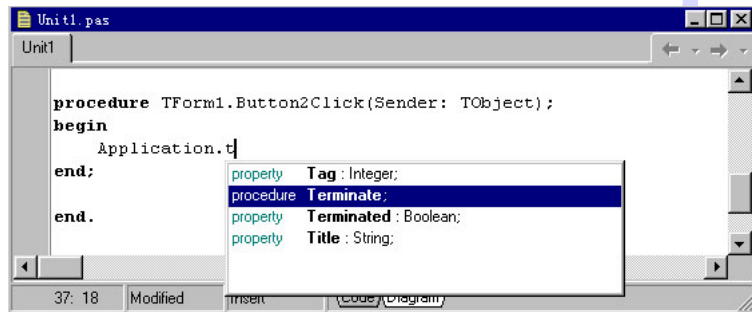


图 1-13 过滤一个对象的过程或属性等

上面添加的代码中，Application 是 Delphi 的一个全局对象，表示一个应用程序。

Terminate 是该对象的一个过程，其作用是关闭整个应用程序。

注意：Delphi 6 提供了对象属性、过程等内容的过滤功能，可以帮助开发人员迅速找到需要的属性等。

以上两个触发事件完整的实现代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage('您输入的内容是：'+Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Application.Terminate;
end;
```

#### 1.4.5 保存文件

用鼠标单击系统菜单 File|Save All 或 File|Save Project As...，Delphi 会弹出一个对话框，要求给单元文件输入一个名字，可输入任何自己喜欢的名字，不过最好是使用一个与实际内容有联系的名字，此处用 u\_test 的名字，并选择相应的保存目录，然后单击“保存”按钮，这样代码编辑器中的代码会以 u\_test.pas 的名字保存到磁盘上，代码编辑器中该单元的标识符也会由 Unit1 变为 u\_test。如果不希望改名，可使用系统缺省的名字 Unit1.pas。

在接着弹出的项目文件名确认对话框中输入 prj\_test，再单击“保存”按钮，整个项目会以 prj\_test.dpr 的名字保存到磁盘上，将来直接打开该项目文件即可打开所有内容。如果用系统的缺省名字，则是 Project1.dpr。

#### 1.4.6 运行程序

用鼠标单击系统菜单 Run|Run，或直接按 F9 键，则整个项目开始运行，如果在建立的应用程序的编辑框中输入“程序测试成功！”，则界面的显示如图 1-14 所示。

在上面执行程序中并没有先编译(Compile)再运行，是因为直接运行程序时会自动进行编译。遇到错误时则会在代码编辑器下面的信息(Message)窗口内显示出来，可在修改错误后重新运行程序直至成功。

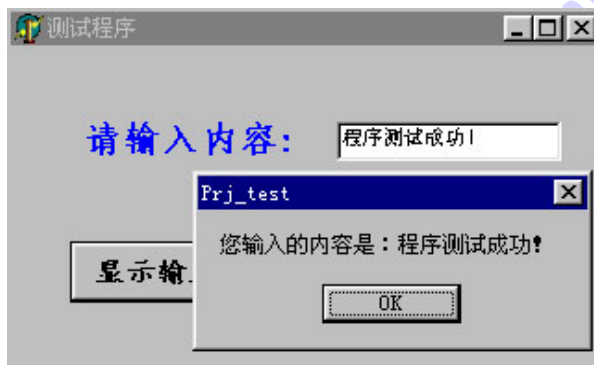


图 1-14 程序运行的结果



## 1.5 获得帮助

Delphi 提供了功能非常强大的帮助系统，可以随时获得需要的帮助信息，下面就列出常用的几种方法：

1. 选择任一个组件，或将光标放置在一个属性或代码编辑器的保留字上，然后按 F1 键，就会显示与当前对象相关的帮助信息。
2. 可以直接通过 Help 菜单获得帮助，既可浏览帮助目录，也可通过“索引”面板搜索需要的帮助信息。
3. 编辑代码时，可获得一个对象的子对象的列表。

## 第2章 开发数据库应用程序

# HOPE



## 声 明

本电子版不包括本章内容，请看配套图书相关章节。

北京希望电子出版社

2001

希望电子出版社

## 第3章 开发 BDE 数据库应用程序

数据库引擎（BDE，Borland Database Engine）是 Delphi、C++Builder、IntraBuilder、Paradox for Windows 和 Visual dBASE 中特有的用于访问数据库的一种机制，它可以让多个应用程序共享。BDE 提供了强大的 API 调用函数库，可以对本地及远程数据库进行操作，并且提供了几乎所有数据库的驱动程序，比如可以访问本地数据库 Paradox、dBASE、FoxPro 和 Access，通过 SQL Links 驱动程序又可以访问远程数据库，比如 InterBase、Oracle、Sybase、Informix、Microsoft SQL server 和 DB2，并且还可以通过 ODBC 适配器来访问数据库。

要使用 BDE 建立数据库应用程序，还必须搞清楚 SQL Links 的作用。Windows 下的 SQL Links 是 Borland 公司产品中提供的用于连接到远程数据库服务器的一系列使用 BDE 的数据库驱动程序，通过建立查询，SQL Links 可以模拟完整的导航能力，并通过使用 Borland 应用程序中一些方便的特点，使用户访问和操纵 SQL 数据库中的数据。SQL Links 产品包中的数据库驱动程序需要安装适当的供应商服务器连接 API 或接口软件。

尽管 BDE 提供了非常强大的数据库访问功能，但是它也有不少缺点，比如建立安装盘时，必须将与 BDE 有关的文件打包进去，这使应用程序的发行盘非常大。同时，通过 BDE 建立数据库应用程序也比较麻烦，因为要访问大部分数据库时，都必须在 BDE 中对该数据库进行必要的配置，并且还必须建立一个数据库别名，数据库应用程序都是通过该别名来访问数据库服务器的。

### 3.1 使用 BDE 连接数据库

#### 3.1.1 BDE 的体系结构

使用 BDE 开发数据库应用程序，也要遵循通常的数据库体系结构。一个 BDE 应用程序除了需要 Delphi 数据库应用程序通用的用户界面元素、数据源和数据集以外，还要包括以下两个方面：

- 一个或多个数据库组件，用来控制事务及管理数据库连接。
- 一个或多个会话组件用来隔离数据访问操作（比如数据库连接），并且可以管理成组的数据库。

BDE 数据库应用程序中各个组件之间的关系如图 3-1 所示。

#### 3.1.2 使用 BDE 管理器

通过从 Windows“开始”菜单中选择 Borland Delphi 6 工作组，再选择 BDE Administrator 选项，如图 3-2 所示；或者在控制面板中选择 BDE 管理器图标，都会打开 BDE 管理器，如图 3-3 所示。要通过 BDE 建立数据库应用程序，就必须在 BDE 管理器中配置这些数据库，并建立相应的数据库别名。应用程序对数据库的访问，都是通过 BDE 别名实现的。

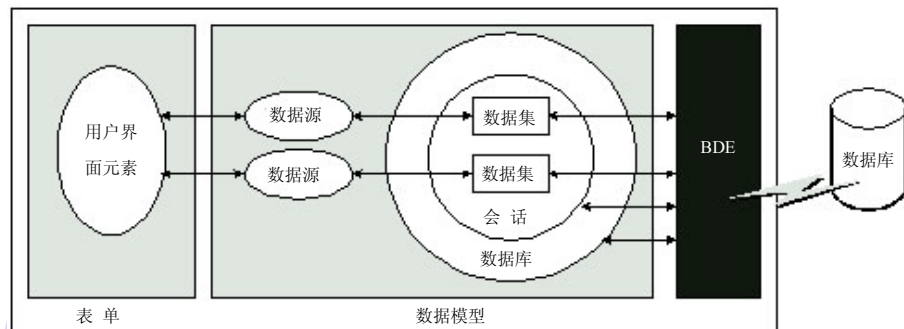


图 3-1 BDE 数据库应用程序中各个组件之间的关系



图 3-2 打开 BDE 管理器的菜单

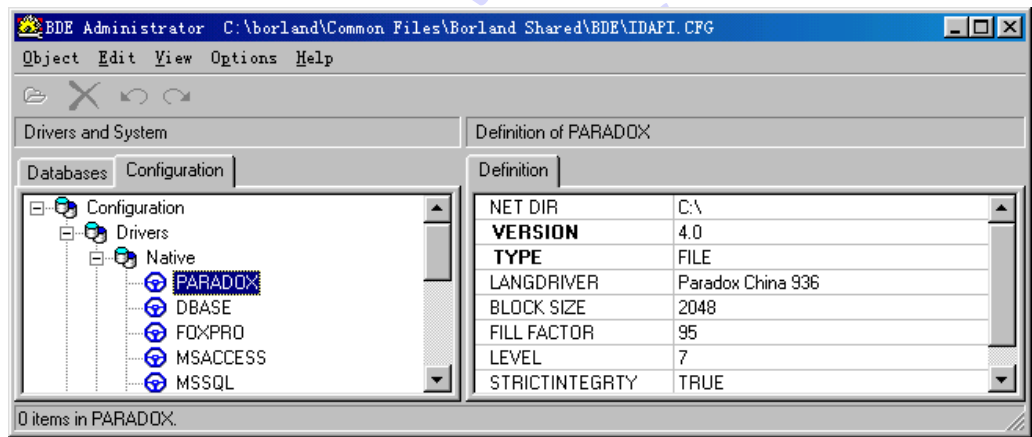


图 3-3 BDE 管理器的管理界面

1. BDE 管理器介绍 BDE 管理器的运行程序是 bdeadmin.exe。BDE 管理器面板包括两部分：左边部分有两个标签页，其中 Configuration 标签页用于配置连接到各种数据库使用的参数，比如使用的 DLL、服务器名等，Databases 标签页用于建立并管理数据库别名，只有配置好一种数据库后才能建立该数据库的别名，否则无法连接到数据库；面板的右边部分是面板左边部分当前选择项的配置细节，所以称为 Definition（定义）面板，该面板的所有配置参数都保存在系统缺省的配置文件 IDAPI32.CFG 及 Windows 的注册表中，具体

位置是: HKEY\_LOCAL\_MACHINE/SOFTWARE/Borland/Database Engine/CONFIGFILE01。

BDE 被设计成面向对象的程序, 可以很容易被扩充或定制, 当需要访问一个附加的数据库时, 只要为该数据库安装合适的 BDE 或 ODBC 驱动即可。

在 Client/Server 环境下, 应用程序和开发工具驻留在客户端的 PC 机上, 而数据源驻留在数据库服务器上, BDE 很完美地适应了 C/S 环境, 因为它对服务器的数据库及本地 PC 机上的数据库都提供了透明的访问。

在 BDE 下安装的数据库驱动大体分为如下三种类型:

- Delphi 标准 (Standard) 驱动程序 比如 Paradox, dBASE, Access, FoxPro 及文本数据库的驱动程序。
- SQL 数据库驱动程序 该驱动程序基于 C/S 结构的 SQL 数据库, 比如 DB2、InterBase、MySQL、Oracle 和 Sybase 等, 这些数据库在客户端计算机上都有各自的 BDE 驱动程序, 这样才能保证数据库的正确连接。

**注意:** 使用 C/S 结构的 SQL 数据库时, 在客户端必须安装 SQL Link 软件, 该软件包提供了 InterBase、DB2、Informix、MySQL、ORACLE、Sybase 和 Microsoft SQL Server 的驱动程序, 它可以通过建立一个查询对象, 完全模拟数据库的数据导航能力。

- ODBC 驱动程序 任何 ODBC 驱动都可用于 BDE, 这是因为 BDE 中有一个 ODBC 的连接通道, 通过 ODBC 驱动, Delphi 应用程序可以访问 ODBC 数据库的数据, 以进行交叉的数据库操作。如果一些 ODBC 驱动没有在配置文件中进行配置, 可通过 BDE 的函数, 比如 DbAddAlias 和 DbOpenDatabase 等, 将 ODBC 驱动自动添加到 BDE 的数据库别名列表中。

2. 配置 BDE 数据库并建立数据库别名 下面通过两个例子, 说明如何通过 BDE 管理器配置不同的 BDE 数据库及建立数据库的别名。

(1) InterBase 数据库 下面说明如何配置 InterBase 数据库及建立一个 InterBase 数据库别名。该数据库的位置是: d:\InterBase\examples\database\employee.gdb。

1) 配置 InterBase 数据库 打开 BDE 管理器, 单击 Configuration 标签页并展开 Drivers\Native 下的所有数据库名称, 然后选择 InterBase, 则 BDE 管理器右边的 Definition 面板会显示出配置 InterBase 数据库使用的所有参数。几个主要参数的配置如下:

- SERVER NAME (服务器的名字) d:\InterBase\examples\database\employee.gdb。
- USER NAME (连接到数据库的用户名) sysdba。

其他参数可以保留默认值, 此时 BDE 管理面板的显示如图 3-4 所示, 在 InterBase 的左边会出现一个绿色的三角图标, 表示该数据库的配置参数进行了修改, 单击工具条上的 Apply 按钮 (蓝色向右的箭头), 保存修改的数据库。

2) 建立 InterBase 数据库的别名 数据库的别名实际上是数据库的一个实例, 由于可以为同一个数据库建立多个数据库实例, 所以也就可以为其建立多个数据库别名。

为前面配置的 InterBase 数据库建立别名的方法如下:

在 BDE 管理器中, 单击 Databases 标签页, 然后在下面的 Databases 上面单击鼠标右键, 则会弹出一个上下文菜单, 如图 3-5 所示。从该菜单中选择 New 选项, 则会打开新建数据库别名对话框, 单击该对话框中的下拉框, 从中选择 INTRBASE, 此时的显示如图 3-6 所示。单击 OK 按钮关闭该对话框, 则会在 BDE 管理器 Databases 树状结构中显示出新建立

的 InterBase 数据库的别名 INTRBASE1，此时可以将该数据库别名修改为其他名字，然后单击工具条上的 Apply 按钮，在弹出的确认对话框中选择 OK 按钮，则会保存该 InterBase 数据库别名的所有设置参数，并完成该别名的建立。

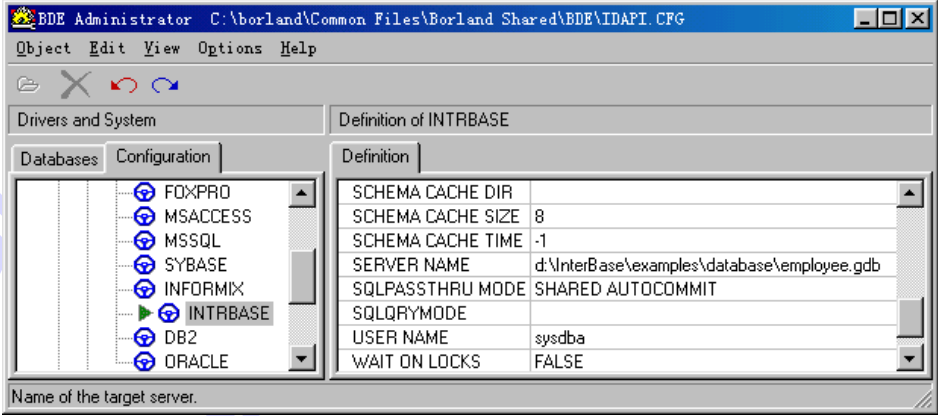


图 3-4 配置 InterBase 数据库的界面



图 3-5 BDE 管理面板的上下文菜单

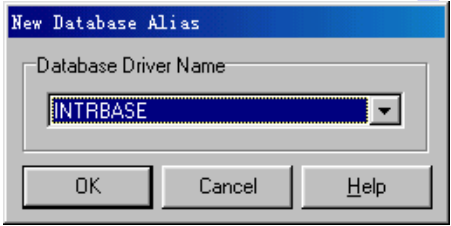


图 3-6 新建立数据库别名对话框

(2) Oracle 数据库的配置 其配置及建立别名的方法与 InterBase 数据库的方法一致，所以下面只说明几个主要的配置参数：

- DLL32 Oracle 数据库使用的动态库，Oracle 7.3 使用 SQLORA32.DLL；Oracle 8i 选择 SQLORA8.DLL。
- VENDOR INIT Oracle 数据库使用的初始库，Oracle 7.3 选择 ORA73.DLL；Oracle



8i 选择 OCI.DLL。

- **SERVER NAME** 设置服务器的名字。由于应用程序的开发一般使用 Oracle 数据库的客户软件，所以要通过 SQL\*NET 建立 Oracle 的实例，而该实例可以在 BDE 管理器的 Databases 标签页 SERVER NAME 选项中选择（会出现一个下拉框，列出 SQL\*NET 建立的实例），所以该参数一般在 Databases 标签页配置，不要在 Configuration 标签页配置。
- **USER NAME** 设置 Oracle 数据库的用户名，比如为 scott。

以上参数配置完后，将修改的内容应用于 Oracle 中。就可以在 Databases 标签页建立 Oracle 数据库的别名。

（3）**ODBC 数据库别名** 如果要通过 BDE 和 ODBC 建立数据库的连接，则需要在 Configuration 标签页设置该数据库对应的 ODBC 参数，大部分数据库使用默认的设置即可。然后在控制面板中建立对应于某个数据库的 DSN，则该 DSN 会自动显示在 BDE 管理面板的 Databases 标签页中。

3. 连接到数据库 下面以连接到 InterBase 数据库为例，说明如何通过 BDE 管理器建立数据库的连接。

在 BDE 管理器 Databases 标签页中选择新建立的 InterBase 数据库别名 INTRBASE1，用鼠标单击该别名左边的加号“+”，则会弹出数据库注册对话框，在 User Name 中输入用户名（比如 sysdba），在 Password 编辑框中输入口令（比如 masterkey），用户名和口令注意区分大小写。然后单击该对话框的 OK 按钮，如果用户名及口令正确，则应该能够连接到数据库。此时 INTRBASE1 左边的图标会发生变化，以黄绿色的方框表示数据库已经连接成功。同时单击左边的加号，在展开的列表中可以看到该数据库中的基表、索引等内容，可以查看这些基表的组成及数据。

4. 使用 SQL Explorer SQL Explorer 的界面与 BDE 管理器非常相似，可用来管理数据库的别名及数据库中的内容。同时，通过该面板可以对数据字典进行管理。另外，对于重复使用的一些字段，或者多个字段使用相同的属性，可以在 Dictionary 的 Attribute Sets 中定义该字段的一个数据集，设置好该字段的属性。然后在数据集的字段管理器（通过字段管理器的 Add All Fields 上下文菜单添加数据集的所有字段）中添加所有字段，再将其中一个字段与 SQL Explorer 中对应的字段（Dictionary 的 Attribute Sets 中定义属性）关联即可。这样当将字段管理器中的该字段拖拉到表单中时，就会利用这些已经设置好的属性。

## 3.2 BDE 组件面板

Delphi 中使用 BDE 建立数据库应用程序，可以通过 BDE 组件面板的各个组件实现。下面首先说明该面板各个组件的作用及主要属性。

BDE 面板提供了三个数据集，分别是：TTable、TQuery 和 TStoredProc，下面分别对这三个数据集的作用进行分析。

### 3.2.1 TTable 组件

该数据集用来表示数据库的一个基表，即它可以代表一个基表中的所有列和行；也可

以代表一个基表中的部分列和行，可以通过设置属性 `Filter` 的值来筛选一个基表中的部分数据。`TTable` 组件提供了非常强大的数据库访问功能，可以访问 BDE 支持的所有数据库。

因为 `TTable` 组件是一个使用 BDE 的数据集，所以它必须与一个数据库组件和一个会话组件 (`TSession`) 相关联，这样就可以通过设置其 `TableName` 属性来决定其对应的基表。`TTable` 组件是唯一可以访问本地一些数据库基表的数据集，这些数据库是 Paradox、dBASE、FoxPro 和使用逗号 “,” 间隔的 ASCII 文本基表等。

1. 指定本地基表的类型 如果一个应用程序要通过 `TTable` 组件访问一个本地数据库，则必须在应用程序中指定要访问的数据库基表的类型，该类型通过 `TTable` 组件的 `TableType` 属性决定，该属性是一个隐含属性，无法通过对象观察器设置它，但是可以通过代码设置。

默认情况下，`TableType` 属性的值是 `ttDefault`，此时 BDE 会根据数据库使用的文件扩展名决定基表的类型，各种文件扩展名与基表类型的对应关系见表 3-1。

表 3-1 文件扩展名与基表类型的对应关系

文件扩展名	基表的类型
没有文件扩展名	Paradox
.DB	Paradox
.DBF	dBASE
.TXT	ASCII text

如果本地的 Paradox、dBASE 和 ASCII 文本基表使用了表 3-1 中列出的文件扩展名，则可以使用 `TableType` 属性的默认值 `ttDefault`；否则，在应用程序中必须设置 `TableType` 属性来指明本地数据库基表的类型。`TableType` 属性可设置的值及这些值对应的基表类型见表 3-2。

表 3-2 `TableType` 属性可设置的值及这些值对应的基表类型

属性值	基表类型
<code>ttDefault</code>	BDE 根据文件扩展名决定基表类型
<code>ttParadox</code>	Paradox
<code>ttDBase</code>	dBASE
<code>ttFoxPro</code>	FoxPro
<code>ttASCII</code>	用逗号分隔的 ASCII 文本基表

2. `TTable` 组件的主要属性

- `Active` 确定是否打开一个数据集，设置为 `True` 则会打开数据集，这样界面上的数据库控件才能显示数据库的数据。如果使用默认值 `False`，则会关闭数据集，也就无法显示数据。该属性与 `TTable` 组件的 `Open` 方法作用相同。打开一个数据集 `Table1`，可使用如下语句：

`Table1.Active:=True;`

- `DatabaseName` 指定与当前数据集关联的数据库的名字。该数据库名字可以是一个 `TDatabase` 组件，也可以是一个 BDE 别名。
- `Exclusive` 如果将该属性设置为 `True`，则当前应用程序会锁定该记录集，其他应

用程序将无法访问其数据；只有在该应用程序结束后，才能释放对数据集的锁定。该属性只能应用于 Paradox 或 dBASE 数据库。例如，将 Table1 锁定的语句如下：

```
Table1.Active := False;
Table1.Exclusive := True;
Table1.Active := True;
```

**注意：**要将一个数据集锁定，必须先关闭该数据集。在设计过程中，如果一个数据集的 Active 属性设置为 True，则不能将 Exclusive 设置为 True，否则会出现异常。

- **Filter** 该属性用于指定数据集的筛选器，即相当于 SQL 语句中的 Where 子句，这样应用程序中只能操作那些符合筛选器设定条件的记录。其中，数据型条件可以直接使用，而字符型条件要使用单引号 “'” 包括起来。多个条件之间使用 and 表示 “与” 的关系，使用 or 表示 “或” 的关系。例如，下面是有 3 个条件的语句：

```
name like ' M%' and age=20 and school= ' 北京大学'
```

**注意：**要使该属性起作用，必须将 Filtered 属性设置为 True。

- **Filtered** 该属性控制 Filter 属性的使用。设置其为 False，则 Filter 属性无法起作用，这是默认的设置值；设置其为 True，则 Filter 属性中设置的值才能使用。
- **FilterOptions** 该选项包括以下两个选项：
  - ◇ **foCaseInsensitive** 该属性决定是否严格按照 Filter 中设置条件的大小写查询内容，设置为 True 则表示按照大小写匹配内容。
  - ◇ **foNoPartialCompare** 该属性决定星号 “\*” 是否作为一个数字通配符对待。设置为 True 表示将其作为单个的字符，而不作为通配符；设置为 False 则当作通配符，可匹配多个数字。

- **IndexFieldNames** 设置按照那些字段进行记录的排序，这些字段必须是关键字段，多个字段之间可以使用逗号 “,” 间隔。
- **IndexFiles** 指定一个或多个 dBASE 的索引文件进行字段排序。注意该属性只能用于 dBASE 数据集。
- **IndexName** 用于设置基表的第二个索引。如果该属性是空的，则一个基表按照默认的索引排序，如果是 dBASE 基表，则按照记录在基表中的物理顺序排序。
- **MasterSource** 该属性只用于建立主从表的应用程序中，设置从属表关联的主表对应的数据源。
- **MasterFields** 用于建立主从表的应用程序，指定与从属表关联的多个主表中的字段，多个字段通过分号 “;” 间隔。该属性只有设置了 MasterSource 属性后才起作用。
- **Name** 设置数据源的名字，默认是 Table1、Table2 等。
- **Readonly** 如果该属性是 True，则设置数据源是只读的，这样就无法对该数据源进行修改、插入及删除数据等操作。
- **SessionName** 设置一个数据库会话的名称。如果没有设置该属性，则使用默认的会话 Session。
- **TableName** 如果已经设置了 DatabaseName 属性，则可以从该属性的下拉框中选择一个基表的名字，则该数据源的数据就来自该基表。

- **TableType** 设置基表的类型，各个属性值的意义见表 3-2。
- **Bof** 该属性返回一个布尔值，表明当前记录是否是数据集的第一条记录。
- **Eof** 该属性返回一个布尔值，表明当前记录是否是数据集的最后条记录。
- **CanModify** 该属性返回一个布尔值，表明一个应用程序是否允许向基表中插入、编辑或删除数据，如果是 **False** 值，表示不允许这些操作。
- **RecordCount** 确定当前该数据集中记录的总数。
- **State** 指明当前数据集的操作状态。可设置的属性值见表 3-3。

表 3-3 数据集操作状态的属性值及其意义

属性值	基表类型
DsInactive	数据集已经关闭，所以数据无法使用
DsBrowse	只能浏览数据而无法修改，这是一个打开的数据集的默认状态
dsEdit	可以修改活动的记录
dsInsert	该活动记录是新插入的，还没有提交到数据库中，可以对该记录继续修改，也可以提交或删除它
dsSetKey	只适用于 TTable 和 TClientDataSet 数据集，表示可以进行记录的搜索，或在进行一个 SetRange 操作，限制的数据可以被查看，但是没有数据可以被编辑或插入
dsCalcFields	正在发生 OnCalcFields 事件。不能编辑非计划的字段，也不能插入新记录
sFilter	正在发生 OnFilterRecord 事件，限制的数据可以被查看，但是没有数据可以被编辑或插入
dsNewValue	当一个字段组件的 NewValue 属性被访问时，内部使用的暂时状态
dsOldValue	当一个字段组件的 OldValue 属性被访问时，内部使用的暂时状态
dsCurValue	当一个字段组件的 CurValue 属性被访问时，内部使用的暂时状态
dsBlockRead	不能修改数据感知控件。当移动到下一条记录时，也无法触发一些事件
dsInternalCalc	当一个字段需要计算的值具有 FieldKind 属性的值为 fkInternalCalc 时，内部使用的暂时状态
dsOpening	正在处理一个打开数据集，但是还没有结束。当打开一个数据集并进行异步提取数据时，会出现该状态

- **Exists** 该属性返回一个布尔值，确定一个基表是否存在。

3. TTable 组件的主要方法

- **ApplyUpdates** 将缓冲区中的数据写入到数据库中。该方法只有将 **CachedUpdates** 属性设置为 **True** 时才有意义。同时要使用 **CommitUpdates** 方法来清除缓冲区内容。注意，使用缓冲区既有优点，也有缺点。优点是减小了网络的流量，提高了效率；缺点是用户已经修改的数据无法立刻被其他用户使用。一般情况下不要使用缓冲区。其语法为：

procedure ApplyUpdates;

- **BatchMove** 将另一个数据集中的多条记录拷贝到当前基表中，并根据选择的模式进行拷贝。其语法为：

function BatchMove(ASource: TBDEDataSet; AMode: TBatchMode):

Longint;

其中参数 **ASource** 表示源数据集；**AMode** 则表示拷贝的模式，可设置的模式及意义，

见表 3-4。

表 3-4 批移动数据时可选择的模式

属性值	基表类型
batAppend	将源基表中的所有记录都追加到目的基表所有记录的后面。此时两个基表中没有重合的关键字段
batAppendUpdate	将源表中的记录追加到目的基表中，如果有相同的记录，则源表中的记录会替代目的基表中的记录。此时目的基表中必须有一个已经定义的索引来匹配记录
batCopy	将源表中的记录和结构拷贝到目的基表中。如果目的基表已经存在，则先删除该基表，并使用源基表的拷贝来代替它
batDelete	删除目的基表中与源基表中所有相同的记录
batUpdate	如果目的基表中有相应的记录，则使用源基表中的记录代替它。目的基表中必须有一个已经定义的索引来匹配记录

- **Cancel** 如果当前记录的修改还没有提交到数据库中，则取消这些修改。无参数。
- **Close** 关闭一个数据集。无参数。
- **CloseDatabase** 关闭一个数据库连接。其语法为：  

```
procedure CloseDatabase(Database: TDatabase);
```
- **Edit** 使数据集处于编辑状态，以便进行修改及删除等操作。无参数。
- **First** 将记录指针移动到数据集的第一条记录。无参数。
- **FreeBookmark** 释放一个书签使用的资源。无参数，其语法为：  

```
function GetBookmark: TBookmark; virtual;
```
- **GetBookmark** 在数据集中当前活动的记录分配一个书签。无参数，其语法为：  

```
function GetBookmark: TBookmark; virtual;
```
- **GotoBookmark** 将记录指针指向有书签的记录，其语法为：  

```
procedure GotoBookmark(Bookmark: TBookmark);
```

例如，下面的例子中使用了与书签有关的方法：

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Bkmark: TBookmark;
    PrevValue: Variant;
begin
    with ClientDataSet1 do
    begin
        { 定义一个书签以便将来可以返回该记录 }
        Bkmark := GetBookmark;
        try
            { 移动到前一条记录 }
            FindPrior;
            { 获得前一条记录中第一个字段的值 }
            PrevValue := Fields[0].Value;
            { 将记录指针移动到以前定义的书签位置，可能移过多条记录 }
            GotoBookmark(Bkmark);
            { 设置书签所在记录第一个字段的值 }
            Fields[0].Value := PrevValue;
            { 释放书签 }
        finally
            FreeBookmark(Bkmark);
        end;
    end;
end;
```



```

        finally
            FreeBookmark(Bkmark);
        end;
    end;
end;

```

- **IsEmpty** 指明当前的数据集是否包含记录，返回值是布尔类型，无参数。
- **Last** 将记录指针移动到数据集的最后一记录。无参数。
- **MoveBy** 移动到与当前记录的相对值决定的另一条记录，比如，**MoveBy(-8)**表示从当前记录开始，向后移动 8 条记录。其语法为：

```
function MoveBy(Distance: Integer): Integer;
```

- **Next** 移动到下一条记录，无参数。
- **Open** 打开一个数据集，无参数。
- **Prior** 移动到前一条记录，无参数。

4. **TTable 组件的事件** TTable 组件的事件可以划分为三种类别，分别以 **After**、**Before** 和 **On** 开头，各种类型的事件的意义如下：

- **After** 表示发生在某种情形出现以后，比如 **AfterInsert** 事件发生在插入记录后；**AfterScroll** 事件发生在滚动记录之后；**AfterClose** 事件发生在关闭数据集之后。
- **Before** 与 **After** 开头的事件相反，该类事件发生在出现某种情形以前，比如 **BeforeInsert** 事件发生在插入记录前；**BeforeScroll** 事件发生在滚动记录之前。
- **On** 该类事件发生在出现某种情形正在出现时，比如 **OnEditError** 事件发生在编辑数据遇到错误时；**OnUpdateRecord** 事件发生在更新记录时。

### 3.2.2 TQuery 组件

该组件可以使用 SQL 语句访问数据库中一个或多个基表的数据，默认情况下其建立的数据是只读的，只有将其 **RequestLive** 属性设置为 **True**，才可以修改基表的数据，这点很重要。

与 TTable 相比，TQuery 组件有以下特点：

- 通过在 SQL 语句中使用 **Join** 连接词，一次可以访问多个基表。
- 自动访问基表中行和列的子集，而不是访问所有的行和列。

TQuery 组件非常适合于开发伸缩性比较强的应用程序，比如使用本地数据库的一个应用程序将来可能要移植到远程数据库，则使用该组件就非常合适。同时它比 TTable 执行效率高。

#### 1. TQuery 组件的主要属性

- **Active** 如果设置为 **True**，则会激活 SQL 语句显示查询到的数据。
- **DataBaseName** 设置使用的数据库的名字，可以是 BDE 别名，也可以是一个 TDatabase 组件的名字。
- **DataSource** 设置来自第二个数据源的参数，该数据源不能使用当前的查询组件。比如当前设置的是 **Query1** 的属性，则可以使用 **Query2** 或 **Table1** 等其他的数据集。该属性的目的是当前数据集的 SQL 语句中使用了一些参数，这些参数是第二个数据源的一些字段。如果当前数据源中字段的值与某个参数（即另一个数据集中的



字段) 的值相同, 则只会查询出与这些值相同的记录。如果当前数据集的 SQL 语句没有使用参数, 或者通过 Params 属性或 ParamByName 方法设置参数, 则不要设置 DataSource 的值。

例如, 当前数据集 Query1 的 DataSource 属性设置值是 DataSource2, 而 DataSource2 对应的 Table1 中有一个字段为 name, 而 Query1 的 SQL 语句如下:

```
SELECT * FROM NAME_LIST dd WHERE (dd.name = :name);
```

上面语句中 dd.name 是 Query1 中的字段, :name 是 Table1 的字段, 查询结果是两个数据源 name 字段值相等的记录。

- Filter 和 Filtered 与 TTable 组件中的对应属性作用相同。由于在 TQuery 的 SQL 语句中可以直接使用 Where 子句, 所以一般不使用这两个属性。

- Params 设置 SQL 语句使用的参数。可以在设计应用程序时设置参数, 也可以在运行时再设置参数。

例如, 数据集 Query1 的 DatabaseName 属性值是 DBDEMOS, SQL 属性内容如下:

```
select * from animals where name like :param1 ;
```

该 SQL 语句中使用了参数: param1 (注意参数前面要使用冒号)。下面通过 Params 属性设置该参数的值。单击该属性右边的按钮, 则会打开参数管理面板, 如图 3-8 所示, 其中显示出了 SQL 语句中使用的参数。

下面设置该参数的初始值。在图 3-7 中选择参数 param1, 然后按 F11 键打开对象观察器, 在 DataType 属性中选择 ftString, 在 Value 属性中输入 %a%, 表示查询 name 字段中含有字母 a 的记录, 此时该参数对应属性面板的显示如图 3-8 所示。这样, 实际运行的 SQL 语句就变为:

```
select * from animals where name like %a% ;
```

现在将 Query1 的 Active 属性设置为 True, 则通过 DBGrid 控件显示数据的设计界面如图 3-9 所示。可见, 只有 4 条符合条件的记录, 这些记录的名称字段中都含有 a。

- SQL 该属性用于设置该数据集使用的 SQL 语句, 单击该属性右边的按钮, 会打开一个编辑框, 可以在该编辑框中输入 SQL 查询语句。该语句可以使用多个参数, 参数前面使用冒号 “:” 标注, 可通过 Params 属性或 ParamByName 方法设置参数值。

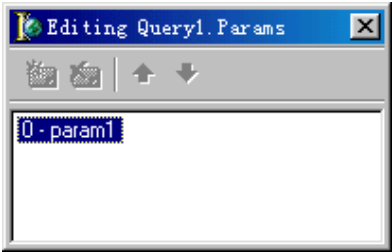


图 3-7 参数管理面板

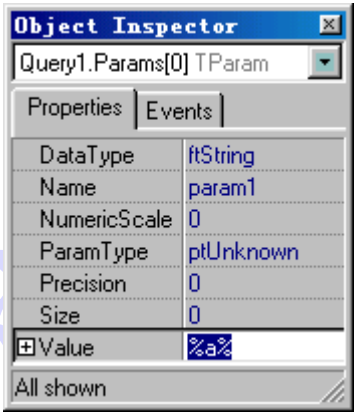


图 3-8 参数 param1 对应属性面板

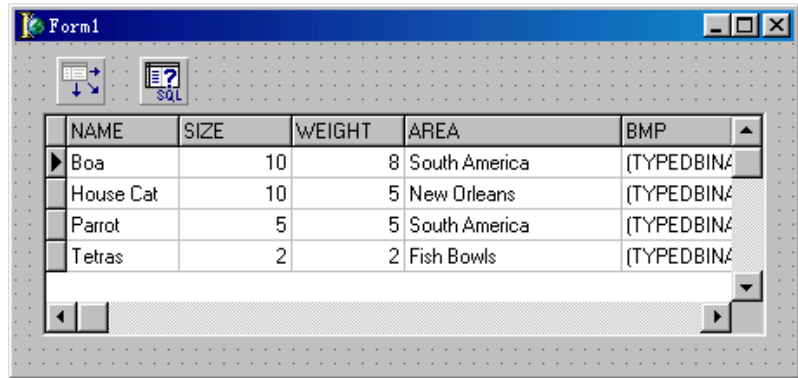


图 3-9 激活数据集 Query1 后的设计界面

- **RequestLive** 该属性决定基表数据是否可以修改，设置为 True 则可以修改。
- **FieldCount** 返回该数据集中字段的数量。
- **Fields** 列出数据集中所有非汇总的字段组件。例如，下面的语句将当前数据集中第三个字段的值在 Edit1 中显示：

```
Edit1.Text := Query1.Fields.Fields[3].AsString;
```

- **FieldValues** 用于访问数据集当前活动记录中所有字段的值。其完整的语法如下：  
property FieldValues[const FieldName: string]: Variant; default;
- 比如，下面的语句表示将编辑框 Edit1.Text 的值赋值给 Name 字段：

```
Query1.FieldValues[' Name '] := Edit1.Text;
```

- **Bof**、**Eof**、**State** 这些属性与 TTable 组件中同名的属性作用相同。

## 2. TQuery 组件的主要方法

- **Append** 在数据集的最后添加一条新的空记录，然后可以通过 FieldValues 属性添加各个字段的值，无参数。
- **Delete** 删除活动记录并将光标移动到下一条记录，无参数。
- **ExecSQL** 运行当前数据集的 SQL 语句，无参数。
- **FieldByName** 通过名字查找一个字段，其语法如下：

```
function FieldByName(const FieldName: string): TField;
```

比如，下面的语句使用 FieldByName 方法将 Edit1 中输入的数值赋值给字段 Size：

```
Query1.FieldByName('size').AsInteger := StrToInt(Edit1.Text);
```

- **Free** 销毁 TQuery 数据集实例并释放其占用的内存，无参数。
- **ParamByName** 按照参数名为参数分配信息。其语法为：

```
function ParamByName(const Value: String): TParam;
```

例如，下面的语句将参数值显示在编辑框 Edit1 中：

```
Edit1.Text := Query1.ParamByName('name').AsString;
```

**注意：**用于 SELECT 中的参数不能为 NULL，但是 UPDATE 和 INSERT 语句中的参数可以为 NULL。

- **Prepare** 在运行一个查询前调用该方法可以提高应用程序的性能，因为它会使远程数据库为该查询分配资源。
- **UnPrepare** 释放 Prepare 方法为查询分配的资源。

- First、Next、Prior、MoveBy、GetBookmark、FreeBookmark、GotoBookmark、Bookmark 等方法与 TTable 中同名方法的作用相同。

### 3. TQuery 组件的事件

与 TTable 中的事件类似，其事件也划分为 Before、After 和 On 开头的事件，分别表示发生在一种情形之前、之后及该情形正在发生。

#### 3.2.3 TStoredProc 组件

该组件在一个基于 BDE 的应用程序中封装了一个存储过程，该存储过程保存在一个数据库服务器上，一个存储过程由一系列 SQL 语句组成，并作为数据库的元数据加以保存（类似基表、索引等）。存储过程用于执行一些需要多次重复并与数据库有关的任务，并将结果传递到客户端。

TStoredProc 组件通过 Params 属性来保持被一个存储过程返回的结果。

注意：有一些数据库不支持存储过程，在使用该数据集时应查看相应的数据库文档。

1. TStoredProc 的属性 与 TTable 和 TQuery 一样，它有许多属性继承自 TDataSet、BDEDataSet、TDBDataSet，所以与 TTable 同名的许多属性具有相同的作用，下面只讲述 TStoredProc 特有的一些属性及继承的主要属性。

- ParamBindMode 决定组件的参数分配到服务器的参数列表中的顺序，有两个选项，其中 pbByName 表示按照名字排列参数，pbByNumber 表示按照序列号排列参数。
- Params 保存一个存储过程的输入和输出参数。其使用方法见前面 TQuery 中同名属性的说明。
- ParamCount 指明存储过程组件中参数的数量。
- Prepared 它是一个 Boolean 类型的属性，决定一个存储过程是否准备运行。
- StoredProcName 表明该对象所在的服务器上存储过程的名字。
- Active 为 True 时则激活该存储过程。
- DatabaseName 设置存储过程所在的数据库。
- Filter 和 Filtered 前者用于设置筛选数据的条件；后者设置为 True 时，才能使用 Filter 中设置的筛选条件。

### 2. TStoredProc 组件主要的方法

- CopyParams 将存储过程的参数拷贝到另一个参数列表中。其语法如下：  

```
procedure CopyParams(Value: TParams);
```
- Create 建立一个存储过程的实例，其语法如下：  

```
constructor Create(AOwner: TComponent);
```
- ExecProc 用于运行服务器上的一个存储过程，没有参数。在调用该方法运行一个保存在服务器上的存储过程以前，应该在 Params 属性中提供一些输入参数，然后调用 Prepare 来绑定这些参数。

例如，下面的代码运行一个存储过程：

```
StoredProc1.Params[0].AsString := Edit1.Text;
StoredProc1.Prepare;
```

```
StoredProc1.ExecProc;
```

- ParamByName 访问基于特定参数名称的参数信息，其语法如下：

```
function ParamByName(const Value: String): TParam;
```

- Prepare 准备运行的存储过程，它会为存储过程分配一些资源。该方法没有参数。
- UnPrepare 释放为前面准备的存储过程分配的资源。

### 3. TStoredProc 组件的事件

TStoredProc 组件的事件与前面数据集使用的事件类似，可以参考 TTable 等组件的事件。

#### 3.2.4 TDatabase 组件

TDatabase 组件用于建立基于 BDE 的数据库连接。开发应用程序遇到下列情形之一时，可以使用 TDatabase 组件建立数据库的连接：

- 永久的数据库连接。
- 定制数据库服务器注册。
- 事务控制。
- 应用程序使用特定的 BDE 别名。

当连接到一个远程的 SQL 数据库服务器时，如果需要使用 BDE 处理数据库事务，就可以使用 TDatabase 组件。

注意：在使用 BDE 中的数据集建立数据库应用程序时，可以使用 TDatabase 组件，也可以不使用该组件建立数据库的连接。

##### 1. TDatabase 组件主要的属性

- AliasName 该属性用于指定一个数据库别名，可以从下拉框中选择。这些数据库别名都是通过 BDE 管理器建立的，每个别名都用于连接到特定的数据库。所以在建立应用程序时，必须通过数据库别名来连接到一种数据库。
- Connected 确定是否激活数据库的连接。如果设置为 True，则表示建立了到数据库的连接。
- DatabaseName 设置与该 TDatabase 组件相关联的数据库的名字，实际上是一个数据库的别名。如果该名字与 BDE 中已经建立的别名相同，则其他 BDE 数据库组件的 AliasName 和 DriverName 属性不需要设置；如果不相同，则必须提供有效的 AliasName 和 DatabaseName 属性，并且还必须提供 DriverName 和 Params 属性。如果要连接到 Paradox 或 dBASE 数据库，可以提供数据库完整的路径名。
- DataSets 提供一个数据库组件所有获得数据集的索引数组。其定义如下：  

```
property DataSets[Index: Integer]: TDBDataSet;
```
- Directory 指定 Paradox 或 dBASE 数据库的工作目录。
- DriverName 指定数据库驱动程序的名字。注意该属性中指定的必须是一个连接到实际数据库的 BDE 的名字，而不是一个数据库的别名。
- Exclusive 如果设置该属性为 True，则一次只能允许一个应用程序单独访问该数据库。
- IsSQLBased 该属性决定一个数据库连接是使用 BDE SQL 链接驱动程序及 ODBC

套接层，还是使用一些 STANDARD（标准）的驱动程序进行数据库通信。如果设置为 True，则使用前者与第三方的 ODBC 驱动程序通信；如果使用 STANDARD 驱动程序，则应将该属性设置为 False。标准的驱动程序包括 Paradox、dBASE 和 ASCII 文本文件。

- **KeepConnection** 该属性指定一个应用程序在所有数据集都关闭的情况下，是否仍保持到该数据库的连接。设置为 True 表示保持。
- **LoginPrompt** 默认情况下，应用程序在打开一个新的数据库连接以前，都要显示一个标准的注册对话框来验证用户的身份。如果希望关闭该对话框，可以将该属性设置为 False。
- **Params** 用于设置与 BDE 别名关联的数据库组件包含的数据库连接参数，可以在该属性打开的对话框的 Key 一栏输入多个参数名字，在 Value 一栏输入对应的参数值。
- **ReadOnly** 如果该属性设置为 True，则指定的数据库连接只能提供只读的访问，不能修改。
- **SessionName** 确定数据库组件使用的会话的名字，一般使用默认的 Session，即设置为 Default。
- **设置多个属性** 在数据模块或表单中新插入的 Tdatabase 上面双击鼠标，则会打开该数据库组件的主要属性设置面板，通过该面板可以设置多个属性。其中 Alias name 和 Driver name 两个选项中只能设置一个选项。在参数列表中，单击 Defaults 按钮可以显示出一些常用参数的设置值，单击 Clear 按钮，则会清除显示的参数。如果选择下面两个复选框，则会显示一个注册提示框，并且不保持活动的连接。该面板的显示如图 3-10 所示。

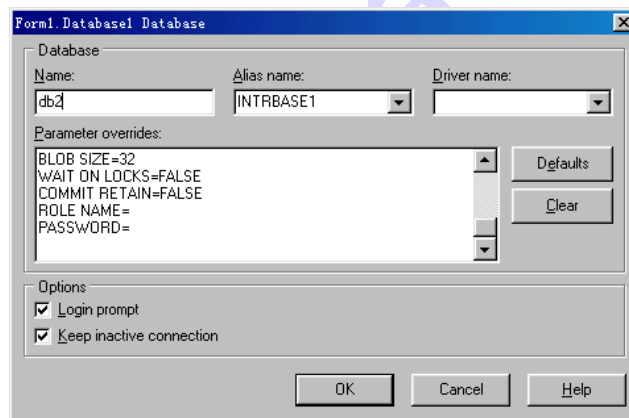


图 3-10 同时设置数据库组件的多个参数

## 2. TDatabase 组件主要的方法

- **ApplyUpdates** 将特定数据集缓存中的数据提交到数据库服务器。语法如下：

```
procedure ApplyUpdates(const DataSets: array of TDBDataSet);
```

例如，Database1.ApplyUpdates([DataSet1, DataSet2])表示将 DataSet1 和 DataSet2 两个数据集保存在缓存中的数据提交到数据库中。



- **CloseDatasets** 该方法会关闭所有的数据集。
- **Commit** 该方法会将当前事务中的所有数据提交到数据库中并长期起作用。
- **Executes** 用于运行一个 SQL 命令。语法如下：  

```
function Execute(const SQL: String; Params: TParams = nil; Cache: Boolean = False; Cursor: phDBICur = nil): Integer;
```

例如: `Database1.Execute(SQLstmt, stmtParams, False, nil);`

该语句中 `SQLstmt` 包含要执行的 SQL 语句; `stmtParams` 包含使用的参数; `False` 表示不使用缓存; `nil` 表示 SQL 语句没有返回结果集。

- **GetFieldNames** 该过程可以获得一个基表中所有字段的名称列表, 其语法如下:  

```
procedure GetFieldNames(const TableName: String; List: TStrings);
```

例如: `Database1.GetFieldNames('Orders', ListBox1.Items);`

该语句表示将 `Orders` 基表中所有的字段显示在 `ListBox1` 中。

- **GetTableNames** 使用该方法可以获得数据库中所有基表的名称列表, 语法如下:  

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

如果 `SystemTables` 参数的值为 `True`, 则表示只显示系统表; 否则, 只显示一般的表。

例如: `Database1.GetTableNames(ListBox1.Items, False);`

该语句表示将数据库中一般的基表显示在 `ListBox1` 中。

- **Rollback** 表示将当前事务中对数据进行的所有修改都取消。
- **StartTransaction** 表示开始一个新的事务。
- **Close** 表示关闭一个数据库连接。
- **Open** 表示打开一个数据库连接。
- **Free** 表示销毁 `TDatabase` 对象并释放其占用的内存。

### 3. TDatabase 组件主要的事件

- **OnLogin** 当一个应用程序连接到数据库时会触发该事件。
- **AfterConnect** 在连接到数据库之后触发该事件。
- **BeforeConnect** 在连接到数据库之前触发该事件。
- **AfterDisconnect** 在断开数据库连接之后触发该事件。
- **BeforeDisconnect** 在断开数据库连接之前触发该事件。

#### 3.2.5 Tsession 组件

**Session** (会话) 是在一个应用程序中同时管理多个数据库连接、驱动程序、游标及查询等内容, 它将各种数据库访问操作隔离开来, 并用于维持应用程序的一些状态。

开发简单的 BDE 数据库应用程序时, 一般不需要使用手工添加或设置一个 BDE 会话, 因为所有的 BDE 数据库应用程序会自动包括一个默认的会话组件, 其名字为 `Session`, 但是在设置一些数据库组件的 `SessionName` 属性时, 却使用 `Default`, 表示使用默认的会话。该会话用于控制应用程序中没有与其他会话关联的所有数据库组件。默认会话在设计阶段既不显示在数据模块中, 也不显示在表单中, 但是可以通过代码访问该会话的一些属性及方法。



由于 BDE 数据库应用程序会自动调用默认会话，所以一般不需要编写任何代码。但是遇到下列情况时，应通过代码调用该默认会话：

- 需要显式激活或不激活一个会话时。
- 需要修改会话的属性。
- 需要运行一个会话的方法来实现一些数据库操作。
- 需要响应一个会话事件。
- 设置 Paradox 目录的位置。
- 管理与使用会话的数据库连接配置或数据集有关的 BDE 别名。

BDE 会话的功能都是通过 TSession 组件实现的。可以通过设置该组件的属性、使用其方法及事件来处理与会话有关的一些操作。它主要用于管理一个应用程序中一组数据库连接。TSession 一般用于标准、Paradox 多网络文件和多线索的数据库应用程序中。默认的会话 Session 只能用于标准的数据库应用程序，并用于处理标准的数据库连接。

数据库应用程序必须同时访问位于网络上不同位置的 Paradox 基表时，应该建立多个会话，每个会话用于一个单独的网络位置。

**注意：**如果一个应用程序使用了多个会话，则可以通过 TsessionList 组件管理它们。默认的会话列表组件称为 Sessions，它不是默认会话 Session 的英语复数形式，而是代表了两种不同的对象。

#### 1. TSession 组件主要的属性

- Active 指定一个会话是否是活动的。设置为 True 则表示为活动的。其默认值是 False。
- AutoSessionName 该属性主要用于多线索的数据库应用程序，指定是否为每个线索自动建立唯一的会话名字。默认值是 False，可用于标准的数据库应用程序。
- DatabaseCount 通过该属性可以获得与当前会话相关的活动数据库组件的数量。
- Databases 用一个索引数组来表示一个会话中所有活动的数据库组件。语法如下：

```
property Databases[Index: Integer]: TDatabase;
```

其中，数据库组件的编号为：DatabaseCount - 1。

- KeepConnections 一般情况下，一个数据库组件到数据库的连接在其所有的数据集都关闭时，也会关闭该连接。如果将该属性设置为 True，则在一个会话中建立的临时数据库连接在其所有数据集关闭后，它也不会断开。
- Locale 确定会话组件使用的 BDE 语言驱动程序。
- NetFileDir 用于设置包含 BDE 网络控制文件 PDOXUSRS.NET 的目录。
- PrivateDir 指定一个目录，用来保存一些临时表处理文件，这些文件由用于与一个会话关联的数据库组件的 BDE 产生。
- SessionName 指定会话的唯一名字。
- SQLHourGlass 如果该属性为 True，则在 BDE 操作期间，鼠标光标会变为沙漏的形状。
- TraceFlags 指定在运行时跟踪 SQL 监视器的数据库操作。各种操作的表示值及意义，见表 3-5。

表 3-5 各种数据库操作的表示值及意义

表示值	意义
tfQPrepare	监视 Prepare 语句
tfQExecute	监视 ExecSQL 语句
tfError	监视服务器错误信息。这些信息可能包括一个错误代码
tfStmnt	监视所有的 SQL 语句
tfConnect	监视数据库连接和断开连接的操作，包括连接句柄的分配和释放连接句柄
tfTransact	监视一些事务语句，比如 StartTransaction、Commit 和 Rollback 语句
tfBlob	监视 blob 数据类型的操作
tfMisc	监视没有被其他标志选项覆盖的语句
tfVendor	监视对数据库服务器的直接 API 函数调用
tfDataIn	监视从服务器接收的数据
tfDataOut	监视发送到服务器的数据

## 2. TSession 组件主要的方法

- **AddAlias** 向会话中添加一个特定的 BDE 别名，该方法只适用于 SQL 数据库服务器。其语法如下：

```
procedure AddAlias(const Name, Driver: String; List: TStrings);
```

其中 Name、Driver 参数分别表示别名的名字及其对应的数据库驱动程序类型，List 则表示别名的列表。

例如：Session1.AddAlias('NewAlias', 'Oracle', List1);

表示将别名 NewAlias 添加到 List1 中，对应的数据库驱动程序类型是 Oracle。

- **AddDriver** 为一个 SQL 数据库服务器向会话中添加一个指定的 BDE 驱动程序。其语法如下：

```
procedure AddDriver(const Name: string; List: TStrings);
```

- **AddPassword** 向当前会话中添加一个密码，用于访问加密的 Paradox 或 dBase 基表。其语法如下：

```
procedure AddPassword(const Password: String);
```

- **AddStandardAlias** 向会话中添加一个标准的 BDE 别名，只用于 Paradox、dBASE 或 ASCII 类型的基表。其语法如下：

```
procedure AddStandardAlias(const Name, Path, DefaultDriver: String);
```

- **Close** 断开会话中所有的数据库连接，并关闭该会话。

- **CloseDatabase** 关闭与当前会话关联的一个数据库连接。其语法如下：

```
procedure CloseDatabase(Database: TDatabase);
```

- **DeleteAlias** 从会话中删除一个别名，其语法如下：

```
procedure DeleteAlias(const Name: String);
```

- **DeleteDriver** 从会话中删除一个指定的 BDE 驱动程序。其语法如下：

```
procedure DeleteDriver(const Name: String);
```

- **DropConnections** 如果一些数据库组件没有被激活，则释放所有这些与会话关联的组件。

- FindDatabase 从一个数据库的会话列表中查找指定的数据库组件，其语法如下：  

```
function FindDatabase(const DatabaseName: String): TDatabase;
```
  - GetAliasDriverName 获得与一个会话关联的、指定 BDE 使用的数据库驱动程序的名字。
  - GetAliasNames 用于提取一个永久的 BDE 别名列表。
  - GetAliasParams 提取与特定 BDE 别名关联的参数。
  - GetConfigParams 提取 BDE 配置信息。
  - GetDatabaseNames 获得一个会话中所有数据库的名字列表。
  - GetDriverNames 获得一个会话使用的所有驱动程序的名称列表。
  - GetDriverParams 获得与一个 BDE 驱动程序关联的参数列表。
  - GetFieldNames 获得一个指定基表中所有字段的列表。
  - GetPassword 调用 OnPassword 事件，并显示默认的口令对话框。
  - GetStoredProcNames 获得在一个 SQL 数据库中保存的所有存储过程的列表。
  - GetTableNames 获得一个数据库中所有基表名称的列表。
  - IsAlias 决定一个字符串值是否与会话已知的 BDE 数据库别名一致。
  - ModifyAlias 添加或修改 BDE 别名参数。
  - ModifyDriver 添加或修改 BDE 驱动程序参数。
  - Open 打开一个会话，并使之成为当前会话。
  - OpenDatabase 打开一个已经存在的数据库或建立一个暂时数据库组件并打开它。
  - RemoveAllPasswords 删除添加到当前会话中用于访问加密的 Paradox 基表的所有口令。
  - RemovePassword 与 RemoveAllPassword 相似，但是一次只删除一个口令。
  - SaveConfigFile 将当前 BDE 的配置信息中任何改变的内容都写入到磁盘上的配置文件中。
3. TSession 组件的属性
- OnPassword 当一个应用程序试图第一次打开一个 Paradox 基表，并且 BDE 有足够的访问权限时触发该事件。
  - OnStartup 当一个应用程序激活一个会话时触发该事件。

### 3.2.6 TBatchMove 组件

该对象用于在一组记录或整个基表上执行数据库的操作，即进行批处理操作。它可以实现如下功能：

- 将一个数据集中的多条记录添加到一个数据库基表中。
  - 从一个数据库基表中删除数据集中的多条记录。
  - 拷贝一个数据集来建立一个新的数据库基表，或者覆盖一个已经存在的基表。
- 以上操作可以通过设置该组件的 Mode 属性来指定。

注意：以前提到过，TTable 通过使用 BatchMove 方法也可以执行同样的批处理操作。

#### 1. TBatchMove 组件主要的属性

- AbortOnKeyViol 该属性决定在出现违反数据完整性或关键值的情况下，一个批

处理操作是否立即停止。

- **AbortOnProblem** 该属性决定当需要截取数据来适应特定的目的时，是否立即终止批处理操作。
- **ChangedTableName** 用于建立一个本地的 Paradox 基表来保存被批操作改变的所有记录的未改变版本。
- **CommitCount** 决定有多少记录在提交发生以前进行了批移动。
- **Destination** 指定一个 TTable 对象，作为批操作的目的地。
- **KeyViolCount** 报告因为数据完整性或关键值等原因没有从目的地中替换、添加或删除的记录的数量。
- **KeyViolTableName** 指定将要建立的 Paradox 基表的名字，用来包含来自源基表的一些记录，这些记录由于违背了数据完整性或关键值等约束而不能分享批操作。
- **Mappings** 指定映射一个批操作的列。
- **Mode** 指定执行 Execute 方法时 TBatchMove 对象进行的操作，可以添加、替换、删除或拷贝记录，可设置的属性值与表 3-4 中列出的值相同。
- **MovedCount** 报告从源基表到目的基表移动的记录数量。
- **ProblemCount** 报告源基表没有移动到目的基表记录的数量。
- **ProblemTableName** 指定一个 Paradox 基表，包含所有没有从源基表移动到目的基表的记录。
- **RecordCount** 指定调用 Execute 时，可以从源基表移动到目的基表的最大数量。
- **Source** 指定作为批操作源的数据集。
- **Transliterate** 当源数据集和目的基表使用不同的语言驱动程序，并且数据中可能包含扩展的 ASCII 字符时，该属性应设置为 True。

## 2. TBatchMove 组件主要的方法

- **Execute** 执行由 Mode 属性决定的批操作。

例如，下面是应用该方法的一个例子：

```
with BatchMove1 do
begin
    Mode := batAppend;
    AbortOnKeyViol := False;

    Execute;

    StatusBar1.SimpleText := '添加的记录数量是: '+IntToStr(MovedCount -
    KeyViolCount);
end;
```

## 3. TBatchMove 组件主要的事件

该组件没有提供任何事件。对象观察器中显示的是目的和源基表的属性。

### 3.2.7 TUpdateSQL 组件

TUpdateSQL 组件提供 SQL 语句用于更新 TQuery 或 TStoredProc 组件表示的只读数据集，但是这些组件的 **CachedUpdate** 属性必须设置为 True。一个只读的数据集可以通过设计或环境决定。如果设计时将数据集设置为只读的，在应用程序的用户接口中不能实现对数

据的更新，但是可以通过程序实现；如果由环境决定一个数据集是只读的，则表示 BDE 只能返回一个只读的结果集，这通常发生在查询多表时，这种查询的只读属性是由 SQL 语句定义的。

在应用程序中，TUpdateSQL 对象一般放置在数据模块或表单中，并通过 TStoredProc 或 TQuery 组件的 UpdateObject 属性链接到这些组件，这样属于这些更新对象的 SQL 语句在应用缓存区的更新时也会自动应用。

1. TUpdateSQL 组件主要的属性

- DatabaseName 指定更新数据提交到的数据库。
- DataSet 确定保持更新数据的数据集。在使用单个更新对象时，当设置源数据集的 UpdateObject 属性时会自动设置该属性，不需要手工设置；但是如果使用了多个更新对象，则必须在运行时，在 BeforeUpdateRecord 或 OnUpdateRecord 事件处理器中设置 DataSet 属性。
- DeleteSQL 指定一个 SQL DELETE 来通过缓存删除一条记录。该语句中可以使用参数：例如：DELETE FROM "Orders.db" WHERE DeptNO = :OLD\_No
- InsertSQL 指定一个 SQL INSERT 语句来通过缓存插入一条记录。该语句也可以使用参数。例如：  

```
INSERT INTO "Country.db" (Name, Capital, Continent)
VALUES (:Name, :Capital, :Continent) WHERE :OLD_Name = "Rangoon"
```
- ModifySQL 指定一个 SQL UPDATE 语句来通过缓存修改一条记录。该语句也可以使用参数。
- Query 当执行特定的更新时，返回查询对象。其语法如下：  

```
property Query[UpdateKind: TUpdateKind]: TQuery;
```

其中，UpdateKind 参数可取值及意义，见表 3-6。

表 3-6 UpdateKind 参数可取值及其意义

UpdateKind 参数可取值	意义
ukDelete	返回执行 DELETE 语句的查询对象 (DeleteSQL)
ukInsert	返回执行 INSERT 语句的查询对象 (InsertSQL)
ukModify	返回执行 UPDATE 语句的查询对象 (ModifySQL)

- SessionName 确定应用更新的会话的名字。
- SQL 当应用缓冲区的更新时，返回一个特定 SQL 语句。其语法如下：  

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

UpdateKind 参数的可取值见表 3-6。

2. TUpdateSQL 组件主要的方法

- Apply 设置特定 SQL 语句类型的参数，并运行结果语句。其语法如下：  

```
procedure Apply(UpdateKind: TUpdateKind);
```
- ExecSQL 运行一个特定类型的 SQL 语句来执行其他只读结果集的更新，此时必须激活缓存更新。其语法如下：  

```
procedure ExecSQL(UpdateKind: TUpdateKind); virtual;
```
- SetParams 在执行一个 SQL 语句前，在该语句中捆绑需要的参数。其语法如下：



```
procedure SetParams(UpdateKind: TUpdateKind); virtual;
```

UpdateKind 参数的可选值见表 3-6，不过在该方法中这些值表示捆绑参数的 SQL 语句类型。

### 3. TUpdateSQL 组件主要的事件

该组件没有事件。

#### 3.2.8 TNestedTable 组件

TNestedTable 组件封装了一个数据集，该数据集嵌套在另一个基表中，并作为该基表的一个字段。它继承了来自 TBDEDataSet 的 BDE 功能，所以要使用 BDE 来访问嵌套基表的数据。一个嵌套的基表提供了基表组件的更多功能。

注意：TNestedTable 只能用于有 BDE 驱动程序的基表的数据集字段。

#### 1. TNestedTable 组件主要的属性

- Active 指定是否激活一个数据集。
- AutoCalcFields 如果该属性设置为 True，则会触发 OnCalcFields 事件，并且计算搜索字段的值。
- CachedUpdates 如果该属性为 True，则会激活数据集的缓存更新。
- DataSetField 指明主表的一个字段，作为该嵌套数据集的关联字段。
- 其他属性 TTable 中的大部分属性也可用于 TNestedTable 组件。

#### 2. TNestedTable 组件主要的方法

- Cancel 如果一些数据还没有提交到数据库，则取消对这些数据进行的修改。
- Close 关闭该组件。
- Free 销毁该组件，并释放其占用的内存。

#### 3. TNestedTable 组件主要的事件

- Before 开头的事件 在调用该方法或出现该情形以前触发该事件。
- After 开头的事件 在调用该方法或出现该情形之后触发该事件。
- On 开头的事件 正在调用该方法或出现该情形时触发该事件。

#### 3.2.9 TBDEClientDataSet 组件

该组件可以缓存更新使用 BDE 提取的数据。它是一个客户数据集，使用内部的 TQuery 和 TDataSetProvider 组件来提取数据及应用更新。它通过使用 BDE 数据集的 CachedUpdates 属性，提供了一种可选择的更新缓存机制。

当使用 BDE 来提取数据时，TBDEClientDataSet 组件应该使用缓存区的更新。使用该组件有以下优点：

- TBDEClientDataSet 组件既可以提取磁盘上数据文件中的数据，也可以提取数据库服务器中的数据，这可以实现“公文包”（briefcase）风格的应用程序。
- 可以利用客户数据集的一些独特特点，比如维持合计并扩展对筛选器的支持。
- 使用不同数据访问机制的数据库应用程序间移植简单，因为 TBDEClientDataSet 与 TSQLClientDataSet 和 TIBClientDataSet 等客户数据集非常相似。



如果 `TBDEClientDataSet` 经过一个提供者连接到一个本地的 `TQuery` 组件，其作用与 `TClientDataSet` 组件非常相似，只是其源数据集和提供者是内置的。它提供了一些 `TQuery` 和 `TDataSetProvider` 组件的属性及事件，以便可以指定提取数据的数据库服务器、提取哪些数据、影响数据包中哪些信息并在更新处理中提供输入。

#### 1. `TBDEClientDataSet` 组件主要的属性

- `Active` 决定是否激活该数据集。
- `Bof` 表示当前记录是第一条记录。
- `CanModify` 如果该属性为 `True`，则可以插入、编辑或删除数据。
- `CommandText` 一般设置一个 SQL 语句来提取数据库的数据。如果使用参数，则应在 `Params` 参数中设置参数值。
- `DBConnection` 确定将数据集连接到一个数据库服务器的 `TDatabase` 组件。
- `DisableStringTrim` 当该属性使用默认值 `False` 时，将记录中的数据提交到数据库，会将字段值后面多余的空格删除。
- `FieldCount` 获得记录的数量。
- `FetchOnDemand` 如果该属性设置为 `True`，则表示客户数据集会根据需要提取附加的数据包。
- `FieldDefs` 指出数据集中定义的字段列表。
- `FileName` 指定保存客户数据集的文件。
- `Filter` 指出数据集的筛选条件。
- `Filtered` 如果该属性设置为 `True`，则会激活 `Filter` 中设定的条件。
- `IndexFieldNames` 如果要按照一个或多个字段排序，则可以在该属性中输入这些字段的名字。
- `IndexName` 从该属性中选择一个已经建立的索引用于该数据集字段数据的排序。
- `MasterFields` 如果当前数据集对应一个基表，则可以在该属性的编辑框中输入主表中关联的一个或多个字段。
- `MasterSource` 该属性用于设置当前数据集对应的主表使用的数据源。
- `Modified` 如果该属性为 `True`，表示可以修改活动记录。
- `Options` 这些选项决定如何提取及使用数据。
- `PacketRecords` 指明一个数据包中记录的数量或类型。-1 表示提取数据集的所有记录；0 表示只提取元数据；大于 0 的数值表示提取记录的具体数量。
- `Params` SQL 语句中使用的参数，用于传递到提供者。
- `ReadOnly` 如果该属性设置为 `True`，表示只能从数据集中读取数据，而不能修改。
- `UpdateMode` 用于确定应用更新的记录的范围。有三个值，其中 `upWhereAll` 表示更新应用到所有的记录；`upWhereChanged` 表示只更新已经修改数据或关键字段的记录；`upWhereKeyOnly` 表示只用于关键字段。

#### 2. `TBDEClientDataSet` 组件主要的方法

- `ApplyUpdates` 从客户数据集向提供者发送所有修改、插入或删除的记录，并写入到数据库中。其语法如下：

```
function ApplyUpdates(MaxErrors: Integer): Integer; virtual;
```

MaxErrors 表示提供者允许出现错误的最大数量，如果该值为-1，则表示没有错误数量的限制。

- Cancel 取消对还没有提交到数据库的记录的改变。
- Close 关闭一个数据集。
- Edit 使数据集中的数据可以编辑。
- Execute 执行提供者数据集的一个 SQL 命令。
- FieldByName 基于一个字段的名称来查找该字段。
- Insert 用于插入一条记录。
- Post 用于提交已经修改的记录。
- Open 打开该数据集。

### 3. TBDEClientDataSet 组件主要的事件

该组件的事件也分为三类，分别以 Before、After 和 On 开头，表示在调用一个方法或出现一个情形之前、之后及正在出现时触发相应事件。

## 3.3 BDE 应用实例

使用 BDE 开发数据库应用程序比使用 ADO、dbExpress、InterBase 等面板的组件开发要复杂一些，因为连接到每一种数据库都需要在 BDE 管理器中配置这些数据库并建立数据库的别名，但是由于 BDE 与 Delphi 结合非常紧密，所以在开发数据库应用程序中仍然占据着很大的比重。下面的数据库应用程序都使用 Delphi 提供的样例数据库别名 DBDEMO，所以都省略了在 BDE 管理器中配置数据库及建立数据库别名的步骤。但是在实际开发中，应根据连接的数据库不同而需要建立不同的数据库别名，建立别名的方法可参考本章前面相关内容的介绍。

### 3.3.1 通过 TTable 的 Filter 属性实现查询

开发 BDE 数据库应用程序一般不需要 TDatabase 组件，可以直接通过数据集连接到数据库中。但是本实例为了说明 TDatabase 的用法，仍使用了一个 TDatabase 组件。另外，由于本实例中只有少量的数据库不可见组件，所以为了简化应用程序，也没有使用 DataModule 来管理这些组件，但是为了使开发的应用程序层次分明，在实际开发中应尽量使用 DataModule 组件来管理不可见的数据库组件。

另外，本实例主要是检验 Filter 和 Filtered 两个属性在筛选数据方面的作用。

该实例的实现步骤如下：

#### 1. 建立应用程序

使用 File|New|Application 菜单新建一个应用程序，此时会建立一个表单 Form1，该表单作为应用程序的主窗口；然后再使用 File|New|Form 添加一个新的表单 Form2，该表单用于输入查询条件。

#### 2. 在 Form1 中添加组件

在 BDE 组件面板上分别选择 TDatabase、TTable，在 Form1 中分别添加一个数据库组件 Database1 和数据集组件 Table1。选择 Data Access 组件面板的 TDataSource，在表单 Form1 中添加一个数据源组件 DataSource1。通过 Data Controls 组件面板，分别在 Form1 中添加一个数据表格 DBGrid1 和数据导航控件 DBNavigator1。最后通过 Standard 组件面板，在表单中添加一个按钮 Button1。

3. 设置 Form1 中各个组件的属性

各个组件的属性设置值见表 3-7。

表 3-7 Form1 中各个组件的属性设置

组件	属性	设置值	说明
Database1	AliasName	DBDEMO	设置使用的数据库别名
	Connected	True	建立到数据库的连接
	DatabaseName	new_db	设置数据库的名字，数据集将使用该名字
	LoginPrompt	False	屏蔽口令注册提示框
Table1	Active	True	激活数据集
	DatabaseName	new_db	设置该数据集使用的数据库
	TableName	items.db	设置数据集使用的基表
DataSource1	DataSet	Table1	设置数据源对应的数据集
DBGrid1	DataSource	DataSource1	设置数据表格使用的数据源
DBNavigator1	DataSource	DataSource1	设置数据导航器使用的数据源
Button1	Caption	查询条件	设置按钮的标题
	Font	楷体/4 号/加粗	设置按钮标题的字体

以上属性设置完毕后，Form1 的设计界面如图 3-11 所示。

4. 在 Form2 中添加组件

打开 Standard 组件面板，然后在 Form2 中插入两个标签 Label1 和 Label2，Label1 作为标题，Label2 对应 OrderNo；再插入一个下拉列表框 ComboBox1，设置查询条件的逻辑运算符；再插入一个文本编辑框 Edit1，用于输入一个数字作为查询的条件。

打开 Additional 组件面板，在 Form2 的底部放置两个按钮，一个用于查询，一个用于取消操作。

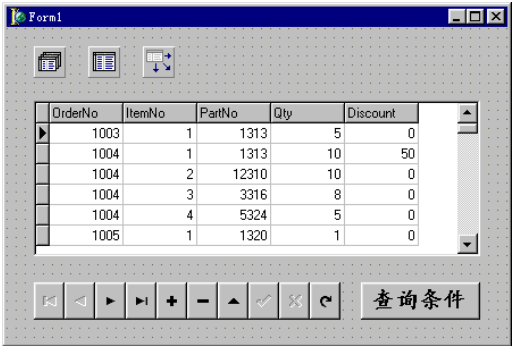


图 3-11 Form1 的设计界面

5. 设置 Form2 中各个组件的属性  
这些组件的属性见表 3-8。

表 3-8 Form2 中组件的属性设置

组件	属性	设置值	说明
Label1	Caption	输入查询条件	提示用户输入查询条件
	Font	楷体4 号 蓝色 加粗	设置标题字体
Label2	Caption	OrderNo	设置 OrderNo 字段的名称
	Font	楷体4 号 加粗	
ComboBox1	Items	> >= = < <=	设置该下拉列表框包含的项
	Text	=	设置默认值
Edit1	Hint	请输入大于 1000 的整数值。	提示用户为 OrderNo 字段输入大于 1000 的整数值作为查询条件
	ShowHint	True	显示 Hint 属性中设置的帮助信息
	Text		删除默认的文本
BitBtn1	Caption	查询	设置该按钮标题
	Glyph		选择查询按钮的图标
	Font	楷体4 号	设置按钮标题字体
	Kind	bkCustom	设置该按钮为自定义类型
BitBtn2	Caption	取消	设置该按钮标题
	Font	楷体4 号	设置按钮标题字体
	Kind	bkCancel	设置按钮为取消按钮

以上所有组件的属性设置完毕后，Form2 的设计界面如图 3-12 所示。

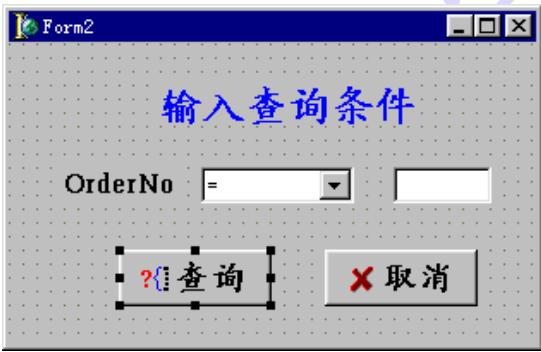


图 3-12 Form2 的设计界面

6. 添加执行代码

(1) Form1 中“查询条件”按钮的代码。该按钮的作用是打开 Form2，让用户输入查询条件。由于 Form1 对应 Unit1 单元，Form2 对应 Unit2 单元，这两个表单及其包含的组件

都在相应的单元中定义，一个表单中的组件要使用另一个表单中的组件，就需要使用 `uses` 语句引用定义了相应组件的单元。

现在 Unit1 中的“查询条件”按钮 Button1 要使用 Form2，则 Unit1 引用 Unit2 的方法如下：先选择 Unit1 或 Form1，然后选择菜单 `File|Use Unit`，此时会弹出一个对话框，如图 3-13 所示，其中列出了当前项目中除了当前选择单元以外的其他单元，这些单元文件都可以被当前选择的单元文件引用。从列表中选择 Unit2，单击 OK 按钮，则在 Unit1 单元文件 `{SR *.dfm}` 语句前面会添加下面一个语句：

```
uses Unit2;
```

现在就可以在 Unit1 中使用 Form2 及其包含的所有组件。

在 Form1 的 Button1 上面双击鼠标，进入 Unit1 代码编辑器，然后在光标位置输入显示 Form2 的代码，该事件的完整代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Form2.ShowModal;
end;
```

`ShowModal` 和 `Show` 方法的作用都是显示一个表单，但是 `ShowModal` 显示的 Form2 表单始终位于应用程序显示窗口的顶层；而 `Show` 显示的 Form2 表单却无法提供该功能，这样当用户单击底层的某个窗口时，比如单击该应用程序的 Form1 时，Form1 就显示在顶层，而表单 Form2 却隐藏到 Form1 的后面，这样用户就无法输入查询条件，从而无法实现我们要求的功能。

(2) Form2 中“查询”按钮的代码。单击该按钮应实现两个功能，一个功能是将本窗口输入的查询条件应用到数据集 Table1 中，查询出符合条件的数据；另一个功能是关闭 Form2。重新将输入焦点移到 Form1 中。由于 Form2 中输入的查询条件实际上被转换为一个字符串来设置 Table1 的 Filter 属性，所以 Unit2 中也要使用下面一条语句来引用 Unit1：

```
uses Unit1;
```

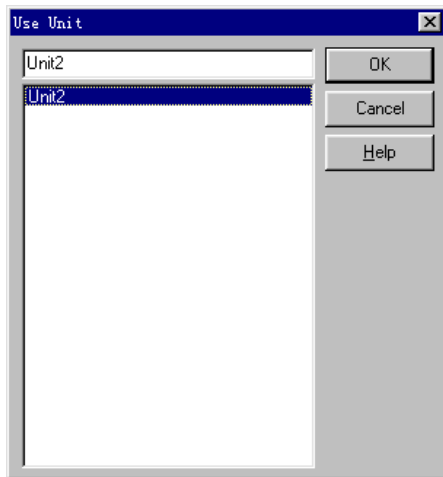


图 3-13 表单 Unit1 引用表单 Unit2

另外，由于用户可能输入不正确的查询条件，比如在 Edit 中输入了非数字内容，此时应用程序就会出现异常。为了避免出现这种异常，应加入异常处理代码，即通过

try...except...end 结构来处理异常,然后在遇到异常时显示一个对话框,提示用户重新输入。

该按钮完整的代码如下:

```
procedure TForm2.BitBtn1Click(Sender: TObject);
begin
    try
        //正常执行的代码
        Form1.Table1.Filter := 'OrderNo'+ ComboBox1.Text+Edit1.Text; //设置
        查询条件
        Form1.Table1.Filtered:=True; //激活查询条件
        close; //关闭Form2
    except
        //处理异常的代码
        on E: Exception do ShowMessage('查询条件错误,请重新输入!');
    end;
end;
```

(3) Form2 中“取消”按钮的代码。该按钮的作用是取消 Form2 中输入的查询条件并关闭窗口。所以只需要使用表单的 Close 方法即可。其代码如下:

```
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
    close;
end;
```

(4) 取消以前输入的查询条件。上面设计的应用程序还存在一个问题,就是只要没有关闭应用程序,则上次在 Form2 中输入的查询条件在重新打开 Form2 时仍然存在。下面通过一个事件来清除这些内容。方法是使用 Form2 的 OnShow 事件,其完整的代码如下:

```
procedure TForm2.FormShow(Sender: TObject);
begin
    ComboBox1.Text:='='; //设置下拉列表框的值为等号
    Edit1.Text := ''; //设置编辑框的值为空
end;
```

## 7. 运行程序

现在运行程序,则会显示出 Form1 窗口,此时会查询出基表 item.db 中所有的数据。单击“查询条件”按钮,则会弹出 Form2,此时可以输入查询条件。如果输入了错误的条件,比如输入了一个字母,则会显示出如图 3-14 所示的对话框,提示用户重新输入。所以在应用程序中对可能出现的异常进行处理,是一种良好的编程习惯,这样可以提供一种人机友好的界面。在窗口中输入正确的查询条件如图 3-15 所示,单击“查询”按钮后,查询的结果如图 3-16 所示。



图 3-14 输入错误的查询条件时弹出的提示对话框





图 3-15 Form2 中输入的查询条件

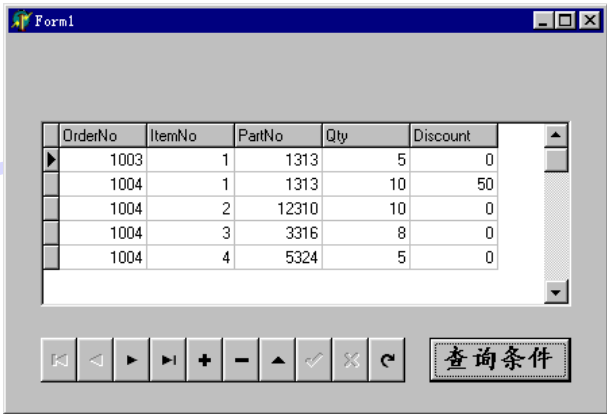


图 3-16 通过输入查询条件查询的结果

3.3.2 通过 TQuery 和 Params 属性实现查询

该应用程序的实现步骤如下：

- 1. 使用 File|New|Application 建立一个应用程序。
- 2. 添加组件。在表单中添加如下组件： BDE 数据集组件 Query1，数据源组件 DataSource1，数据表格组件 DBGrid1，数据导航条 DBNavigator1。
- 3. 设置组件的属性。各个组件的属性设置见表 3-9。

表 3-9 Form1 中各个组件的属性设置

组件	属性	设置值	说明
Query1	Active	True	激活数据集
	DatabaseName	DBDEMOS	设置该数据集使用的数据库
	Params	Param1	该参数的设置见表 3-10
	SQL	select * from items where OrderNo=:param1	设置该数据集的 SQL 语句，使用一个参数 param1 作为查询条件
DataSource1	DataSet	Query1	设置数据源对应的数据集
DBGrid1	DataSource	DataSource1	设置数据表格使用的数据源
DBNavigator1	DataSource	DataSource1	设置数据导航器使用的数据源

表 3-10 参数 param1 的属性设置

参数	属性	设置值	说明
param1	DataType	ftInteger	设置参数的类型为整数型
	Value	1004	设置参数的初始值

以上所有组件的属性设置完毕后，Form1 的设计界面如图 3-17 所示。然后可以运行该应用程序，查询的结果与图 3-17 相似。

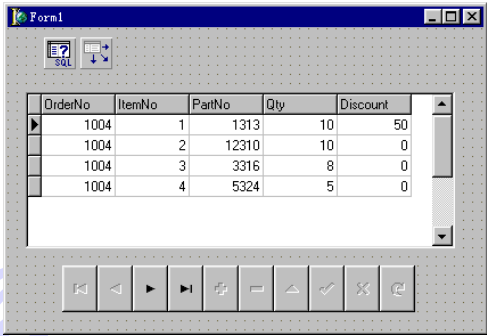


图 3-17 Form1 的最终设计界面

3.3.3 使用 TStoredProc 组件

该实例使用 Delphi 提供的 IBLocal 数据库，该数据库实际上是一个本地的 InterBase 数据库。该数据库有一个存储过程是 ADD\_EMP\_PROJ，用于在向 employee\_project 基表中插入一条记录。可以通过 SQL Explorer 打开该存储过程来查看其定义，如图 3-18 所示。

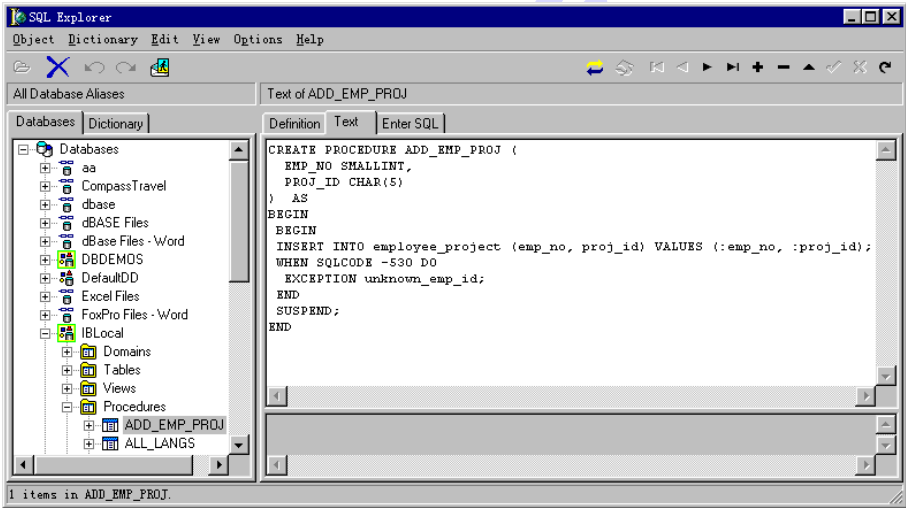


图 3-18 通过 SQL Explorer 查看存储过程 ADD\_EMP\_PROJ 的定义

从图 3-18 中可以看出，该存储过程中定义了两个参数 emp\_no 和 proj\_id，用来输入 employee\_project 基表两个字段的內容。在该实例中，就是通过一个文本编辑框和一个下拉框完成这两个参数的输入，然后运行存储过程将数据插入到基表中。

该实例的实现步骤如下：

1. 通过 File|New|Application 新建立一个应用程序。

2. 添加组件

在表单中分别放置两个面板 Panel1 和 Panel2, 其中 Panel1 用于放置存储过程的参数输入项及执行存储过程的按钮, Panel2 用于显示基表的部分数据, 以便比较存储过程执行前后的结果。将两个面板的 Caption 属性值设置为空值, 再将下面的 Panel2 的 BevelOuter 属性设置为 bvLowered, 这样可以使该面板下陷。

在 Panel1 上添加如下组件: 添加一个存储过程组名 StoredProc1; 添加 3 个标签 Label1、Label2、Label3; 添加一个文本编辑框 Edit1, 一个下拉框 ComboBox1 和一个按钮 Button1。

在 Panel2 上添加如下组件: 添加一个数据表格组件 DBGrid1; 一个数据导航组件 DBNavigator1; 一个基表组件 Table1 和一个数据源组件 DataSource1。

最后, 在 Panel2 上面添加一个标签组件 Label4。

3. 设置各个组件的属性

各个组件属性的设置值见表 3-11。

表 3-11 设置 Form1 中各个组件属性

组件	属性	设置值	说明
Label1	Caption	输入数据	
	Font	楷体4号 蓝色 粗体	
Label2	Caption	EMP_NO:	
Label3	Caption	PROJ_ID:	
Label4	Caption	检验存储过程的执行结果	
	Font	楷体4号 蓝色 粗体	
Edit1	Text		使文本输入框默认值为空
ComboBox1	Items	VBASE MAPDB GUIDE MKTPR DGPII	列出 PROJ_ID 字段的可选值, 该字段的值只能从列表中选择, 不能输入其他内容
Button1	Caption	插入数据	
	Font	楷体4号 粗体	
StoredProc1	DatabaseName	IBLocal	设置存储过程组件使用的数据库
	StoredProcName	ADD_EMP_PROJ	设置已经定义的存储过程
Table1	Active	True	
	Filter	EMP_NO<=8	设置查询条件
	Filtered	True	
	DatabaseName	IBLocal	
	TableName	EMPLOYEE_PROJECT	
DataSource1	DataSet	Table1	设置使用的数据集
DBGrid1	DataSource	DataSource1	
DBNavigator1	DataSource	DataSource1	

以上所有组件的属性设置完毕后，Form1 的显示如图 3-19 所示。



图 3-19 表单 Form1 的设计界面

4. 为 Button1 添加代码

在 Button1 上双击鼠标，打开代码编辑器，然后输入相应的代码，该事件处理过程的完整代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    StoredProc1.Params[0].AsString:=Edit1.Text;           //设置第一个参数
    emp_no的值
    StoredProc1.Params[1].AsString:=ComboBox1.Text;      //设置第二个参数
    proj_id的值
    StoredProc1.Prepare;                                   //为存储过程的运行准备资源
    StoredProc1.ExecProc;                                  //执行存储过程，插入数据
    Table1.Refresh;                                        //刷新DBGrid1中显示的数据
end;
```

5. 运行程序

现在运行程序，在注册窗口的口令编辑框中输入 masterkey，则会打开应用程序的主界面。然后在编辑框 Edit1 中输入 5，从下拉框 ComboBox1 中选择 MAPDB，再单击“插入数据”按钮，则该数据会插入到基表中，此时下面会显示出新插入的记录，如图 3-20 所示。



图 3-20 通过存储过程插入新数据后的显示

## 第 4 章 开发 ADO 数据库应用程序

数据库方面的应用历来是 Delphi 的优势，像基于自动化办公、生产流程控制、局域网以及 Intranet/Internet。例如实时控制，学校与图书馆管理、酒店、商场、银行等管理系统。Delphi 6 不但可以通过 IDE 建立数据库应用，而且可以通过许多其他方法建立基于互联网的 Web 数据库共享与应用。这需要使用 ADO 组件，它不像 IDE，它可以不同的数据库类型进行相应的配置，也无需将一些专用的动态链接库打包，就很容易开发出针对网络数据库的应用软件；很像我们在 WWW 上碰到的 Web 交互应用那样。

应用程序不直接访问物理数据库，而是通过面向用户的接口——数据库引擎。在 Delphi 里有两种数据库引擎，即 Borland 的 BDE 和微软的 ODBC。

BDE 提供一种 32 位的基于 Win 32 的数据库引擎，它可以访问多种数据库。BDE 的主要特点是：具有存取各种主流数据库的统一接口，直接访问数据源；对 Paradox 和 dBASE 数据库提供优先引擎；支持基于 C/S 模式的数据库应用；集成的数据库引擎支持不同数据之间的查询、复制等操作；支持 SQL 并可访问基于 SQL Server 的数据库；支持多线程。

但是，BDE 仅适用于 Borland 系列的产品。

微软的 ODBC 则是针对众多数据库产品，如 Oracl、SysBase、DB2 等，提供了一种统一访问的方案。ODBC 内含的驱动程序与具体数据库相关，一个驱动就是一个支持 ODBC 调用的函数模块。用户通过调用驱动程序提供的接口函数来访问数据库。要使用不同类型的数据库，只需要调整 ODBC 数据库连接。

继 ODBC 的跨平台访问数据的优势，出现了 COM 技术，它利用面向对象的设计思想，在 Internet 的环境下实现统一的访问 OLE DB 技术。这种技术不仅支持关系型也支持非关系模数据库，还支持电子邮件、文件系统和图像数据，以及用户自定义数据对象等多种数据源。继 OLE DB 技术之后，微软又提出了 ADO 技术。ADO 技术提供了一个一致的、高性能的、高兼容性的数据访问接口。它既能实现数据库前端的创建，又能实现中间层的设计。ADO 的应用范围涵盖了从一层到多层数据库应用的解决方案，以及基于 Web 的数据驱动解决方案。并且，ADO 提供了一个比 OLE DB 更容易使用的接口，优化的数据访问手段，减少了网络负载和应用程序前端、数据源之间的层次，所有这些使得 ADO 具有轻巧、高性能的特点。因此，ADO 技术业已成为现今数据库应用开发的一种潮流，并代表了未来技术的发展方向。

ADO 另外一个重要的优势是将被内置在微软的所有操作系统里，包括 Windows 2000，这就意味着使用 ADO 访问数据库不需要在每一台 PC 中再安装 ADO。

### 4.1 ADO 组 件

#### 4.1.1 ADO 简介

ADO 是 Microsoft ActiveX Data Objects 的缩写，是微软开发的只适用于 Windows 操作

系统的数据库访问对象，它由一系列 COM 对象组成，通过 OLE DB 数据提供者访问数据库中的数据。ADO 提供了基于对象且便于使用的接口，它只占用系统很少的资源，但是运行效率却比较高，另外，它还允许用户使用数据库中的各种系统资源，它建立在 OLE DB 技术的基础上，OLE DB 可以对底层数据库数据进行访问。

在实际应用中，用户可能使用不同的数据源存储数据，比如使用 Oracle、SQL Server、Access 等数据库、使用 Excel 电子表格、使用邮件系统或文件系统等，这些数据源的每一种都有自己专用的数据存储格式和存储方法。而要使一个应用程序访问这些不同类型的数据源中的数据，则需要使用一种通用的方法，ADO 就是这样一种通用的方法，因为只要是 ODBC 能够存取的数据源都是 ADO 可以存取的对象。

要使用 ADO 访问数据库，还需要理解与 ADO 有关的一些概念，这些概念包括 ODBC、DAO、OLE DB 等，下面分别进行说明。

- ODBC (Open Database Connectivity, 开放的数据库连接) ODBC 是微软和一些数据库厂商联合制定的，它通过应用程序接口 (API)，提供了一种跨平台的、用来访问关系数据库中数据的手段，是访问数据库的通用方法。
- DAO (Data Access Object, 数据访问对象) DAO 是最早出现在微软的 Access 数据库中，它通过有严格层次关系的对象来操作数据。因为 DAO 是建立在 ODBC 之上的，所以使用 DAO 编写的应用程序可以与多种数据源进行交互。DAO 在访问微软数据库时速度比 ADO 快，而访问其他类型数据库时则性能较差。因为遭到大量的批评，微软不再发布其新的版本。
- OLE DB 微软开发的 OLE DB 应该是 ODBC 的升级产品。它增强了通过 Internet 访问数据的能力，可以对诸如邮件系统等非关系数据库型数据源进行访问，同时还提供了一个基于组件对象模型 (COM) 的数据访问接口。OLE DB 定义了一套 COM 接口，用于数据访问，而不必了解数据的存储方式。ODBC 与 OLE DB 的最大区别是 ODBC 可以跨平台，而 OLE DB 只能在支持组件对象模型 (COM) 的系统平台上使用，即主要在 Windows 平台上使用。OLE DB 中提供了一个针对 ODBC 的数据提供程序，从而可以继续支持已经存在的 ODBC 数据库驱动。

Delphi 中的 ADO 组件封装了 ADO 对象，可用于建立数据库应用程序。基于 ADO 的应用程序的 ADO 层包括 ADO 2.1、一个 OLE DB provider 或 ODBC 驱动程序、特定数据库系统使用的客户软件、访问应用程序的一个数据库后端系统和一个数据库等。

## 4.2 ADO 面板

ADO 组件面板提供了多个组件，通过这些组件可以连接到一个 ADO 数据源、运行多个命令并从基表中提取数据等。要使用这些组件，则运行应用程序的计算机上必须安装 ADO 2.1 以上版本、特定数据库系统使用的客户软件以及 OLE DB 或 ODBC 驱动程序。ADO、OLE DB 和 ODBC 等都可以从微软的站点中免费下载，Windows 系列的操作系统中大部分也都提供了这些功能或驱动程序。而特定数据库的客户软件则需要购买其供应商的产品，比如开发 Oracle 数据库应用程序时就需要安装 SQL\*NET 及 Oracle 客户端软件。

ADO 组件面板提供的组件与 BDE 组件面板提供的组件基本可以实现类似的功能，并



且许多组件也有类似的名字、属性、方法及事件，所以只要掌握了 BDE 组件的使用方法，再学习 ADO 组件时就会比较轻松。

ADO 组件面板工提供了 7 个组件，分别是 TADOConnection、TADODataSet、TADOTable、TADOQuery、TADOStoredProc、TADOCommand 和 RDSConnection，其中后面的 6 个组件可以直接连接到数据库，也可以通过 TADOConnection 组件连接到数据库。这些组件的作用如下：

- TADOConnection 该组件用于建立数据库的连接，ADO 的数据源组件和命令组件可以通过该连接组件运行命令及从数据库中提取数据等。
- TADODataSet 这是 ADO 提取及操作数据库数据的主要数据集，在 BDE 中没有对应的组件。该组件可以从一个或多个基表中提取数据。
- TADOTable 主要用于操作和提取单个基表的数据。与 TADODataSet 类似，它可以直接连接到数据库，也可以通过 TADOConnection 组件连接到数据库。
- TADOQuery 该组件是通过 SQL 语句实现对数据库数据的提取及操作，它可以直接运行数据定义语言（DDL），比如 CREATE、ALTER、DROP 等 SQL 命令。其连接数据库的方式与前两种数据集一样。
- TADOStoredProc 该数据集是专门用于运行数据库中的存储过程的，这些存储过程可能会提取数据，也可能不提取数据。
- TADOCommand 该组件用于运行一些 SQL 命令，这些命令没有数据集返回。所以该组件不是一个数据集组件。该组件可以与支持数据集的组件一起使用，也可以直接从一个基表中提取一个数据集。
- RDSConnection 一个进程或一台计算机传递到另一个进程或计算机的数据集合。

利用 ADO 访问数据的一般进程为：首先向应用程序添加一个 ADOConnection 组件，用于建立与数据库的连接；然后使用一个 ADOConnection 组件或者 ADOQuery 组件向数据库执行 SQL 命令；最后通过数据集获得数据。这时，数据集组件必须将 Connection 属性指向所使用的 ADO Connection 组件，以下将对各个 ADO 组件逐一介绍。

#### 4.2.1 TADOConnection 组件

该组件用于建立数据库的连接，该连接可被多个数据集所共享，但是并不是应用程序中必须的，因为 ADO 数据集及命令组件通过设置其 ConnectionString 属性，可以直接连接到数据库。但是如果多个数据集使用相同的数据库连接时，则使用 TADOConnection 就有一定的优势，因为不必为每个数据集都单独建立数据库的连接，同时也减少了资源的消耗，并且可以建立跨越多个数据集的事务。一个事务（transaction）是数据库操作的一个阶段，用户对数据库的修改都保存在本地计算机的内存中，只有提交一个事务后，才能将修改的内容提交到数据库中；如果选择了回滚事务，则所有的修改将被取消，而不会提交到数据库中。

TADOConnection 组件提供如下功能：

- 控制数据库的连接。
- 控制服务器的注册。
- 管理事务。

- 为关联的数据集提供数据库连接。
- 将 SQL 命令发送到数据库中。
- 获得数据库的元数据 (metadata)。

■ ADOConnection 的常用属性

1. Attributes

此属性用于设置连接的数据库的自动处理事务的行为。它是TXactAttributes类型的集合，包括两个集合元素：

- (1) xaCommitRetaining : 提交一个事务后自动开始一个新的事务。
- (2) xaAbortRetaining : 回退一个事务的同时将开始一个新的事务。

2. CommandTimeout

连接超时属性。用于设置一个命令执行时所能等待的最大时间值，以秒为计量单位。缺省值为 30 秒，即连接命令等待了 30 秒之后还没有被执行，系统就放弃这个命令。

3. Connected

标识和数据库的连接是否处于激活状态。

用户可以查询 Connected 属性的值来判断数据库的连接状态。如果该属性为 true，则表明数据库处于连接状态；为 false，则当前数据库连接关闭。

4. ConnectionString

连接字符串用于指定数据库的连接信息。连接串的标准调用方式为：

```
ADOConnection1.ConnectionString := 'Provider=ProviderRef;Remote  
Server=ServerRef ';
```

其中，连接串支持的常用参数见表 4-1。

表 4-1 数据库连接参数及说明

参数	说明
Provider	数据提供者名称，例如 MSDASQL.1
Password	登录数据库的口令
Persist Security Info	支持安全登录
User ID	登录数据库用户
Data Source	数据源名称。数据源的设置需要额外的操作

假定用户已经有设置好了的 ODBC 数据源（稍后讨论这方面的问题）。双击 ADOConnection 控件或 ConnectionString 属性框，显示如图 4-1 所示的 ConnectionString 设置对话框。

单击“Build”按钮，弹出设置数据连接属性的对话框（图 4-2），是一种类似向导的界面，帮助用户一步一步地设置连接字符串的值。

选择“下一步”，进入设置“连接”页，如图 4-3 所示。

单击“使用数据源名称”的下拉列表，如果有刚刚建立的数据源没有出现在列表里，点击“刷新”按钮即可。

选择“RegTest”数据源。为方便起见，选中“空白密码”和“允许保存密码”。设置完毕，

单击“测试连接”，弹出“测试连接成功”的消息框之后，证明我们的操作成功。

选择“确定”即可完成设置。

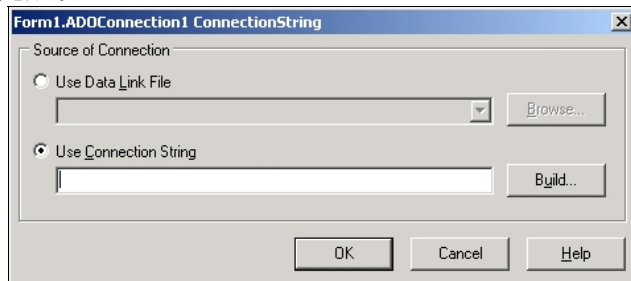


图 4-1 ConnectionString 设置对话框

至于“高级”页是关于数据在存取权限是细微设置，具体的可参看相关手册。“所有”页则是对前面三个页面设置的值的一个摘要，可以对设置的值进行手工调整。如图 4-4 所示。

用户可以根据自己的喜好进行选择操作。

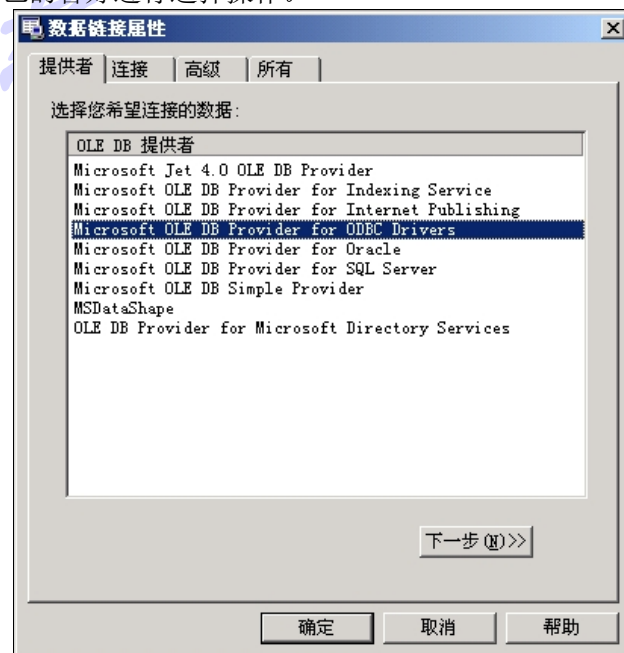


图 4-2 选择数据连接“提供者”的对话框

设置成功之后，用户就可以将属性 Connected 设置为 True，如果没有任何提示信息，说明数据库已经成功连接。此时，属性 DefaultDatabase 就被赋值为连接所指定的数据库的路径。

### 5. ConnectOptions

指定数据库连接是按照同步方式还是异步方式连接。类型 TConnectOption 包含两个值：

- coConnectUnspecified：数据库连接采用同步方式连接。
- coAsyncConnect：异步方式连接数据库。当服务器负载很重的时候，这种连接方

式很有用。引用这种连接方式，在第一次建立连接的时候，应用程序不能获得全部所需的数据。

## 6. CursorLocation

指定数据库指针是指向客户端还是服务器端。类型 `TCursorLocation` 包含两个值：

- **clUseServer**：使用服务器端的数据库指针，适用于数据量大的数据集。
- **clUseClient**：使用客户端的数据指针的时候，数据将被下载到本地计算机上，并在本地进行操作。

## 7. DefaultDatabase

表明数据源成功连接后，数据库的路径。这是由数据源自动赋值的。

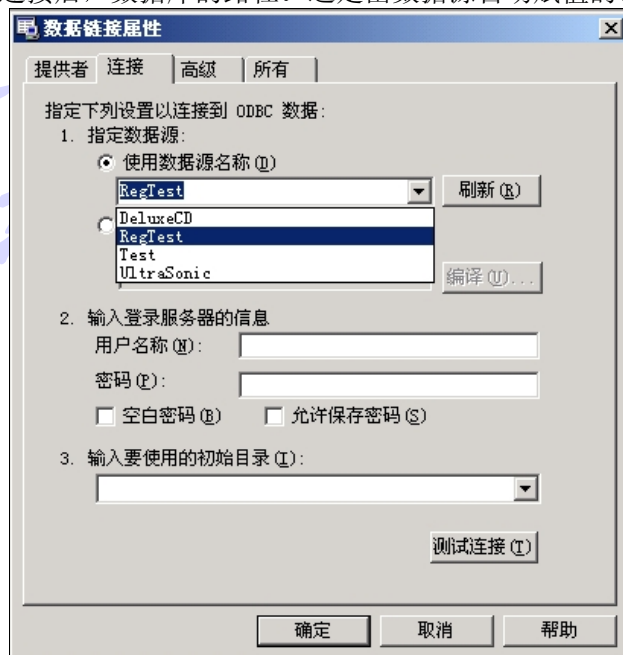


图 4-3 选择数据源

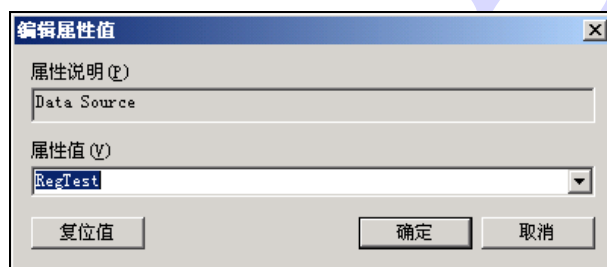


图 4-4 手工编辑数据源连接属性

## 8. IsolationLevel

指定不同事务之间的相互独立的级别。事务实际上是对数据库的一系列操作的集合，事务具有整体性，如果事务中的某一个步骤不能被正确执行，则整个事务都不会被执行。由于数据库服务器可以同时支持多个连接，来自不同连接的事务有可能在同一时刻对同一

个数据进行操作，这就有可能造成数据的不一致性，为防止这种情况的出现，ADO 引入了事务独立级来确定不同事务之间的相互关系。设定事务独立级之后并调用 BeginTrans 方法后，新的事务独立级别将生效。TIsoationLevel 共包含表 4-2 的 9 种常量值。

表 4-2 TIsoationLevel 常量及说明

常量参数	说明
ilUnspecified	使用默认的独立级别，没有其它指定的独立级别
ilChaos	来自更高独立级别的事务对数据的改变不能被当前的事务覆盖
ilReadUncommitted	当前事务可以读取其他事务未提交的数据
ilBrowse	当前事务可以读取其他事务未提交的数据
ilCursorStability	事务提交后数据才能被读取
ilReadCommitted	事务提交后数据才能被读取
ilRepeatableRead	不能读取其他事务的数据，执行 Requery 操作可以获得这些数据
ilSerializable	从其他事务种获取事务的独立级别
ilIsolated	从其他事务种获取事务的独立级别

这些常量的定义在 Microsoft Data Access SDK 中有详尽的说明，需要进一步了解可以查询微软的 SDK 文档。

#### 9. KeepConnection

指定如果在没有打开数据集的情况下是否仍然保持数据库的连接。

频繁地打开和关闭数据库的操作将会影响系统的性能，特别在网络上，会在一定的程度上会增加网络的负载。这个属性设置数据源始终处于连接状态，可以显著提高程序的性能。

#### 10. LoginPrompt

指定在每次建立连接时，是否弹出登录对话框提示用户登录。如果设为 False，则必须在 ConnectionString 中指定登录数据库的用户名和密码。

#### 11. Mode

指定连接对数据库的操作权限。这种连接模式的值如表 4-3 所示。

表 4-3 连接模式参数及说明

参数	说明
cmUnknown	未指定数据库操作权限或无法确定
cmRead	对数据库只能读操作
cmWrite	对数据库只能写操作
cmReadWrite	对数据库可读写操作
cmShareDenyRead	禁止其他用户对数据库读操作
cmShareDenyWrite	禁止其他用户对数据库写操作
cmShareExclusive	禁止其他用户对打开数据连接
cmShareDenyNone	禁止其他用户对数据库任何操作

■ 注册 ODBC 数据源

实现数据源的设置有两种方法：一是利用 Windows 的数据源（ODBC）工具设置，再一个是自己手工编写设置代码。我们一一介绍。

（1）使用 Windows 的数据源（ODBC）工具

在 Windows9X/XP 下，可以在控制面板中直接找到这个工具；Windows 2000 下，可以在控制面板的管理工具找到这个工具。

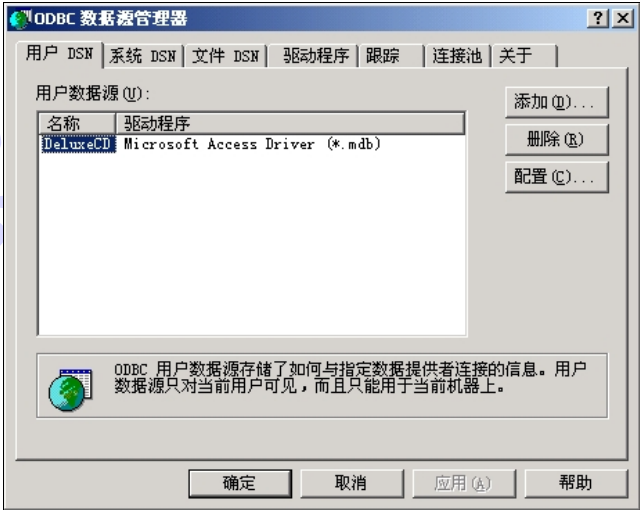


图 4-5 ODBC 数据源管理器

选择“数据源（ODBC）”，运行之后，显示如图 4-5 所示“ODBC 数据源管理器”对话框。



图 4-6 选择数据库驱动程序

选择“系统 DSN”页，然后单击“添加”按钮，弹出数据库驱动程序对话框，提示用户选择连接数据源的的驱动程序。我们以 Microsoft Access 的数据库为例，选择与之相配合数据库驱动程序，如图 4-6 所示。

选取 Microsoft Access Driver(\*.mdb)之后，单击完成。弹出 Access 数据库驱动程序的设置窗口，设定数据源名，然后单击“选择”按钮，弹出“选择数据库”对话框，提示用户选



择数据源要连接的 MDB 数据库。操作界面如图 4-7 所示。

选择“确定”，完成选择数据库。结果如图 4-8 所示。

在图 7-8 中选择“确定”，则数据源设置完毕。结果如图 4-9 所示。

这样，我们就可以在 ADO 组件中引用这个设置好的名为“Test”的数据源。

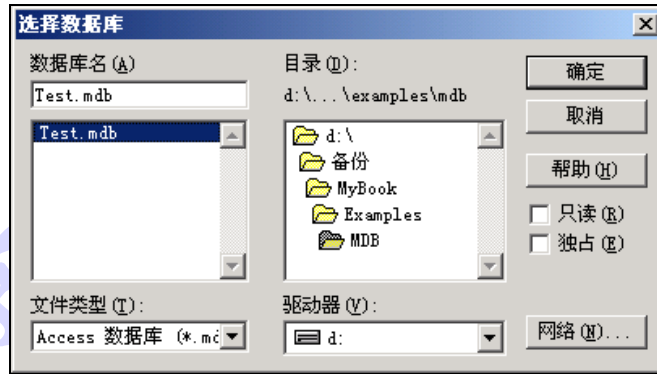


图 4-7 选择数据源连接的 MDB 数据库

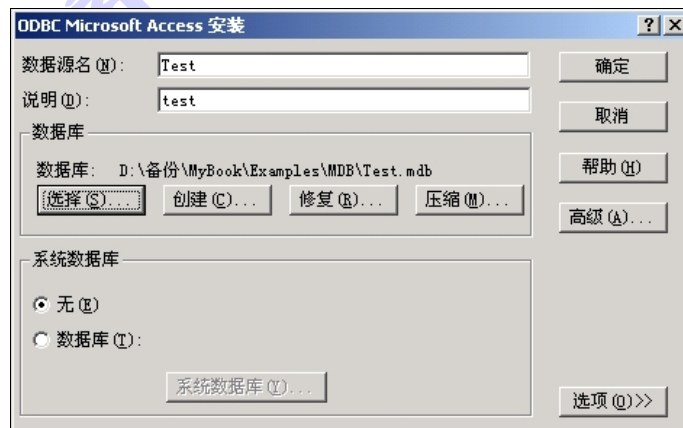


图 4-8 选择完数据库

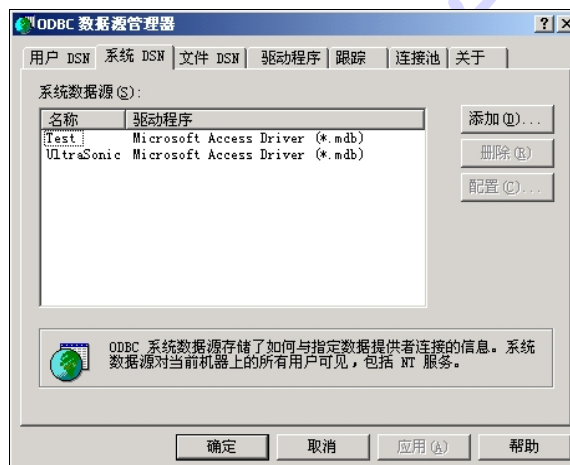


图 4-9 成功设置“Test”数据源

(2) 事实上，仔细观察在注册表中 ODBC 的注册项目，刚刚所做的一些操作都记录在注册表中。我们可以发现刚才添加的数据源项目，如图 4-10 所示。

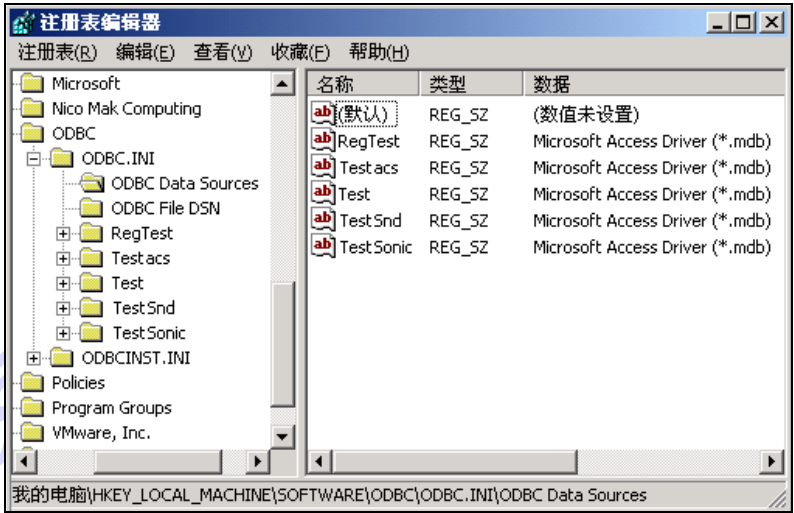


图 4-10 注册表中记录的数据源信息

因此可以专门编写一个注册函数过程来实现前述的一系列操作过程。实现代码如下：

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Buttons, Registry, DB, ADODB;

type
  TForm1 = class(TForm)
    SpeedButton1: TSpeedButton;
    ADOConnection1: TADOConnection;
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

const
  ProMsg='测试注册ODBC数据源';

implementation

{$R *.dfm}
```

```

procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  wReg :TRegistry;
  tmpPath :array[0..255] of Char;
  SysPath :string;
  tmpDBPath :string;
  tPath :string;
begin
  GetSystemDirectory(@tmpPath,255);
  sysPath :=StrPas(tmpPath);

  tPath :=ExtractFilePath(Application.Exename);
  tmpDBPath :=tPath+'..\MDB\Test.mdb'; //指定连接数据库的路径

  if not FileExists(tmpDBPath) then
  begin
    MessageBox(handle,pChar('找不到存储在 '+tmpDBPath+' 的数据文件!'),
      ProMsg,mb_ok+mb_IconWarning);
    Exit;
  end;

  //建立一个Registry实例
  wReg := TRegistry.Create;
  //==注册ODBC==
  with wReg do
  begin
    RootKey:=HKEY_LOCAL_MACHINE;
    if OpenKey('Software\ODBC\ODBC.INI\ODBC Data Sources',True) then
      WriteString('RegTest', 'Microsoft Access Driver (*.mdb)')
    else
    begin
      Application.MessageBox('ODBC 初始化错误!',proMsg,
        mb_ok+MB_ICONEXCLAMATION);
      exit;
    end;
    CloseKey;

    //写入DSN配置信息
    if OpenKey('Software\ODBC\ODBC.INI\RegTest',True) then
    begin
      WriteString('DBQ',tmpDBPath); //数据库目录
      WriteString('Description','Pacs数据源'); //数据源描述
      WriteString('Driver',SysPath+'\odbcjt32.dll'); //驱动程序DLL文件
      WriteInteger('DriverId',25 );
      WriteInteger('SafeTransaction', 0); //支持的事务操作数目
      WriteString('UID', ''); //用户名称
    end
    else
    begin

```

```
Application.MessageBox('ODBC 初始化错误!',proMsg,
    mb_ok+MB_ICONEXCLAMATION);
exit;
end;
CloseKey;

//写入DSN数据库引擎配置信息
if OpenKey('Software\ODBC\ODBC.INI\RegTest\Engines\Jet',True) then
begin
    WriteString('ImplicitCommitSync', 'Yes');
    WriteInteger('MaxBufferSize',2048); //缓冲区大小
    WriteInteger('PageTimeout',10); //页超时
    WriteInteger('Threads',3); //支持的线程数目
    WriteString('UserCommitSync','Yes');
end
else
begin//创建键值失败
    Application.MessageBox('ODBC 初始化错误!',proMsg,mb_ok+
        MB_ICONEXCLAMATION);
    exit;
end;
CloseKey;
Free;
end;
Application.MessageBox('用户信息注册成功!',proMsg,
    mb_ok+mb_iconinformation);
end;

end.
```

为了检验是否成功注册,运行这段程序,然后在查看 Windows 的 ODBC 数据源管理器,我们这段代码手工设置的 ODBC 数据源已经成功的加入,如图 4-11 所示。

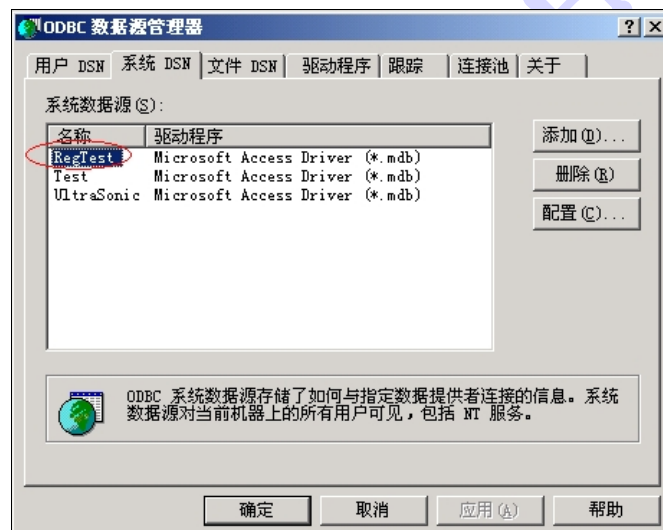


图 4-11 程序自动注册的 ODBC 数据源

### ■ ADOConnection 的主要方法

#### 1. BeginTrans

开始启动一个新的事务。必须保证数据连接处于激活状态。

#### 2. Cancel

关闭与数据库的连接

#### 3. CommitTrans

向数据库提交一个事务。提交成功后，在事务中对数据库所作的修改则写入数据库中，同时一个事务也结束。

4. Execute(const CommandText: WideString; var RecordsAffected: Integer; ExecuteOptions: TExecuteOptions = [eoExecuteNoRecords]);

执行一个 CommandText 类型的 SQL 命令。其中，CommandText 是指定的 SQL 命令；RecordsAffected 指定该命令设计的记录数目；ExecuteOptions 指定命令特征如表 4-4 所示。

表 4-4 ExecuteOption 的值及说明

参数	说明
eoAsyncExecute	异步执行指定的命令
eoAsyncFetch	给定了 Cache 属性的值之后，再异步地取得数据
eoAsyncFetchNonBlocking	非阻塞式线程执行
eoExecuteNoRecords	没有返回记录

#### 5. GetProcedureNames(List: TStrings);

获取数据库服务器上的存储过程名称，获取的存储过程名保存在 List 参数中。

#### 6. GetTableNames(List: TStrings; SystemTables: Boolean = False);

获取数据库中的数据表，获取的表名存放在 List 参数中。SystemTables 参数指示是否获取数据库系统表的名称。数据库系统表是指在数据库中关于数据库数据类型定义和用户信息的数据表，这种系统表是数据库本身自动生成的。

#### 7. Open(const UserID: WideString; const Password: WideString);

打开一个连接。参数 UserID 是数据库用户的用户名，Password 是用户登录数据库的密码。

#### 8. RollbackTrans

撤回一个没有全部执行的事务。事务撤回之后，事务中所作的任何修改都不会写入数据库。

### ■ ADOConnection 的主要事件

表 4-5 ADOConnection 组件的主要事件

事件	说明
AfterConnect	发生在一个连接建立后
AfterDisconnect	发生在断开连接后
BeforeConnect	发生在连接建立前
BeforeDisconnect	发生在断开连接前
OnBeginTransComplete	发生在开始一个事务时

(续表)

事件	说明
OnCommitTransComplete	发生在提交事务成功时
OnConnectComplete	发生在连接完成时
OnDisconnect	发生在断开时
OnExecuteComplete	发生在一个命令执行后
OnInfoMessage	发生在收到数据库的消息
OnLogin	发生在用户登录数据库的时候
OnRollbackTransComplete	发生在一个事务撤回之后
OnWillConnect	发生在发出一个连接数据请求的时候
OnWillExecute	发生在数据库收到一个 SQL 命令并将要执行的之前

4.2.2 ADOCommand 组件

ADOCommand 向数据库发送 SQL 指令并返回请求的数据集。

TADOCommand 组件主要用于运行一些数据定义语言 (DDL) 的 SQL 命令, 或者运行一个没有返回结果集的存储过程。对于返回结果集的 SQL 语句, 则最好使用 TADODataSet、TADOQuery 或 TADOStoredProc 组件。尽管 TADOCommand 组件的 Execute 方法可以返回一个结果集, 但却是通过另一个 ADO 数据集组件来使用该记录集。

TADOCommand 组件与 ADO Command 对象相似, 所以 TADOCommand 组件中的属性和方法在 ADO Command 对象中都能找到相同的名字, 并且具有相同的作用。如果使用 ASP 开发过动态网页, 对此一定会有深刻的认识。

TADOCommand 代表了 ADOCommand (ADO 命令) 对象, 它通过一个 ADO 提供者访问数据库。TADOCommand 组件执行的是其 CommandText 属性中设置的命令, 通过调用 Execute 方法执行该命令。如果该命令中需要使用参数, 则通过 Parameters 属性设置, 该属性与 BDE 数据集 TQuery 组件的 Params 属性的作用及设置方法相同。

■ ADOCommand 的主要属性

1. CommandText

指定要执行的 SQL 命令。可以手工编写, 也可以利用如图 4-12 所示的 CommandText 编辑器对话框来设置这个属性。

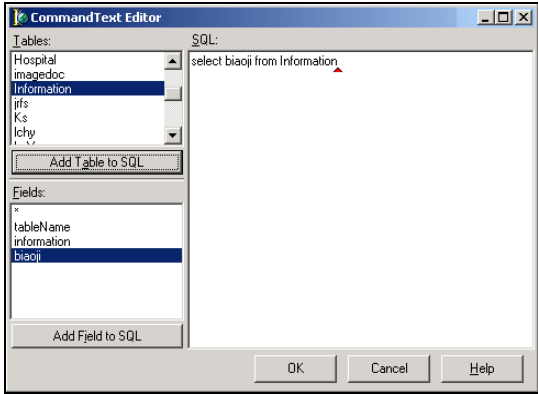


图 4-12 编写 SQL 的 CommandText 编辑器



CommandText 编辑器是专门用来为 ADO 组件来编写 SQL 命令的。Table 列表框用来列出数据库中所有的表，选中一个表，单击“ADD Table to SQL”按钮，CommandText 编辑器就会自动把表名插入 SQL 命令的相应位置。在选中某个表的同时，这个表中的所有字段都会自动地列在 Fields 列表框那。同样，选中 Fields 列表框中的一个字段，单击“Add Fields to SQL”，字段就会插入到 SQL 命令中。操作结果如上图所示。

## 2. CommandType

指定要执行的命令的种类。它可以是表 4-6 中所列值的一种。

表 4-6 CommandType 的参数及说明

参数	说明
cmdUnknown	未知的命令类型
cmdText	文本类型
cmdTable	命令中指定的是一个表的名称
cmdStoredProc	命令中指定的是一个存储过程的名称
cmdFile	命令中指定的是保存数据集的文件名
cmdTableDirect	命令中指定的是表的名称，并返回所有的列

## 3. Connection

指定所使用的数据源连接组件的名称，即 ADOConnection 组件的名称。通过这个属性使得 ADOCommand 能与数据库连接起来。

## 4. ParamCheck

指定在 SQL 命令动态改变的时候，是否需要重置参数列表。

## 5. Parameters

执行 SQL 命令时要用到的参数。在参数查询中，即在 SQL 命令中或在存储过程中需要传递参数的时候才需要设置这个值。并且，在命令类型 CommandType 指定为 cmdText 或 cmdStoredProc 时，参数才有效。

### ■ ADOCommand 的主要方法

#### 1. Cancel

中止一个正在执行的命令。

#### 2. Assign(Source: TPersistent);

把另一个 ADOCommand 组件的所有属性复制到当前的 ADOCommand 组件中。调用的时候，按名存取组件对象。

#### 3. Execute

执行 ADOCommand 组件所包含的命令。返回结果是一个数据记录集，可以被其他 ADO 组件的 RecordSet 记录集属性调用。

### 4.2.3 ADODataset 组件

ADODataset 是 ADO 组件中最通用的一个组件，它能获取并代表任何其他 ADO 组件从数据库返回的数据集合，数据集是通过 SQL 命令返回的从一个表或者是多个表数据。

ADO 组件面板已提供了 TADOTable、TADOQuery 和 TADOStoredProc 等数据集组件，为何又提供了一个 TADODataset 数据集组件？实际上，TADODataset 是一个通用的数据集组件，可以代替其他三个数据集，只要分别将其 CommandType 属性设置为 cmdTable、cmdText 或 cmdStoreProc，并分别在其 CommandText 属性中设置一个基表、一个 SQL 命令或一个存储过程即可。

要使 TADODataset 数据集组件能够正常地发挥作用，则应首先设置其 Connection 或ConnectionString 属性来建立起到数据库的连接。如果要使用一个 RDS DataSpace 对象将该数据集连接到一个基于 ADO 的应用程序服务器，则需要将 RDSConnection 属性设置为一个 TRDSConnection 对象。

由于 TADODataset 组件必须返回一个结果集，所以其 CommandText 属性中如果使用语句，则只能使用 SELECT 语句，而不能使用一引起数据操作语言 (DML)，比如 DELETE、INSERT 和 UPDATE 语句。

同时由于该数据集可以使用 SQL 语句，所以可以从一个或多个基表中查询数据。

#### ■ ADODataSet 的常用属性

##### 1. Active

指示当前的记录集是否处于打开状态。调用 Open 方法，打开数据库，Active 值为 True；调用 Close 方法，数据库关闭，则 Active 为 False。

只有 Active 的值为 False 时，应用程序才能对数据库进行读写操作。需要将 Active 变为 true 的几种情况是：

- (1) 数据集的状态设为 dsBrowse（浏览模式，查看、扫描数据）。
- (2) 如果程序中加载了 BeforeOpen 事件，在该事件被触发时。
- (3) 如果程序中加载了 AfterOpen 事件，在该事件被触发时。
- (4) 通过记录集打开一个数据指针时。

在更改数据集属性，且这些属性影响到数据库的状态或数据显示组件的状态时，要提前将数据集的 Active 属性设置为 False。

##### 2. AutoCalcFields

设为 True，则允许应用程序触发 OnCalcFields 事件。计算字段依赖于当前记录的一个或多个字段，通过已有的字段数据统计计算。该值为 True，在记录数据被修改或者编辑时，就触发 OnCalcFields 事件，应用程序自动更新计算字段的值，以保证数据的一致性。触发 OnCalcFields 事件的条件是：

- (1) 数据集组件的状态变为 dsEdit。
- (2) 记录已经被修改。
- (3) 应用程序从数据库中重新获得一条记录。

可以看出，上述的情况并不都需要更新计算字段。如果用户需要频繁地修改数据，则 OnCalcFields 事件就会不断地调用。频繁地调用在一定程度影响了应用程序的性能。在这种情况下，可以将属性 AutoCalcFields 设为 False。

##### 3. CacheSize

指定数据集的缓冲区大小。数据集首先把数据从数据库中取出，然后保存在内存的一

块区域中，这块内存区域就是所谓的缓冲区。如果设置 `CacheSize` 属性值为 20，则表示数据集将一次从数据库中提取 20 条记录并将这 20 条记录保存在缓冲去中。缺省值为 1，也是其最小值。

4. `CommandText`

指定数据集中所包含的命令，可以是SQL语句、一个表名或者一个存储过程名。常用的调用形式为：

```
with ADODataset1 do begin
  CommandType := cmdText;
  CommandText := 'SELECT * FROM CustomerTable';
  Open;
end;
```

5. `Filer`

数据集的过滤器。通过使用过滤器可以把那些不需要的数据过滤掉。设置 `Filter` 属性的典型方法为：

```
with ADODataset1 do begin
  Filtered := False;
  Filter := 'State = ' + QuotedStr('CA') + ' OR ' +
    'State = ' + QuotedStr('CA');
  Filtered := True;
end;
```

如果用户修改了带有过滤器数据集的数据，修改的结果如果不满足过滤器的条件，则修改的数据就自动从当前的数据集中消失。

6. `MaxRecords`

返回数据集中记录数。默认值为 0。

■ `ADODataset` 的常用方法

1. `GetIndexNames(List: TStringList)`

查询表中的全部索引，返回值将保存在参数 `List` 中。调用方法为：

```
ADODataset1.GetIndexNames(ListBox1.Items);
```

2. `DeleteRecords(AffectRecords: TAffectRecords = arAll);`

删除记录集中的记录。参数 `AffectRecords` 用于指定要删除的具体记录，它的取值可以为表 4-7 所示。

表 4-7 `AffectRecords` 的参数及说明

参数值	说明
<code>arCurrent</code>	仅删除当前记录
<code>arFiltered</code>	删除满足过滤器过滤条件的数据
<code>arAll</code>	删除记录集中的所有记录
<code>arAllChapters</code>	删除 ADO 连接数据部分的全部记录

缺省值为 `arAll`，删除当前记录集中的全部数据。

3. `Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions)`

定位一条记录并把这条记录作为当前记录。其中，`KeyFields` 是索引的字段名；`KeyValues`

是要查找的值；Options 是定位数据的选项，它的值可以是：

- loCaseInsensitive ：定位数据不区分大小写。
- loPartialKey ：部分匹配定位查找数据。例如：查找“书”的时候，会定位查找到“计算机书籍”、“文学书籍”，甚至“书记”。

4. Requery(Options: TExecuteOptions = [])

刷新数据集中的数据。它是通过重新执行原来的命令或 SQL 语句来重新生成记录集。

5. SaveToFile(const FileName: String = ""; Format: TPersistFormat = pfADTG)

把当前数据集中的数据按照指定的格式保存到指定的文件中。FileName 为指定的文件名；Format 为保存的文件格式，它可以选取下列值：

- pfADTG ：按照 ADTG（Advanced Data Tablegram）格式生成文件。
- pfXML ：按照 XML 格式保存文件（需要 ADO 2.1 或更高版本 支持）。

详尽的技术细节可以参考 Microsoft Data Access SDK 技术文档。

6. Seek(const KeyValues: Variant; SeekOption: TSeekOption = soFirstEQ)

搜索记录并移动数据集的指针。搜索动作是以当前数据集中的索引为搜索依据。其中，KeyValues 为被搜索的值。

```
Success := ADODataset1.Seek('Jones', soFirstEQ);
```

如果搜索到复合索引的值，则返回 True，反之为 False。也可以同时搜索多个值：

```
ADODataset1.Seek(VarArrayOf([90030, 90020]), soFirstEQ);
```

这就需要利用函数 VarArrayOf()构造一个数组参数传递给 KeyValues。

Seek 的第二个参数 SeekOption 限定了搜索行为的动作，可以取如表 4-8 所列出的值。

表 4-8 SeekOptions 的取值及说明

参数值	说明
soFirstEQ Record	数据库指针定位在第一条匹配的记录处，如果没有任何匹配记录则指向数据库的末尾记录
soLastEQ Record	数据库指针定位在最后一匹配的记录处，如果没有任何匹配记录则指向数据库的末尾记录
soAfterEQ Record	如果搜索到匹配记录，在指向匹配记录的下一条；如果没有找到，则指向最近似匹配的记录上
soAfter	指向匹配记录的下一条
soBeforeEQ	如果搜索到匹配记录，在指向匹配记录的前一条，如果没有找到，则指向最近似匹配的记录上
soBefore	指向匹配记录的前一条

#### 4.2.4 ADOTable 组件

ADODataset 组件、ADOTable 组件、ADOQuery 组件和 ADOSToreProc 组件都是继承自父类 TCustomADODataset。所以，在属性、事件及方法上有许多共通的地方。

TADOTable 组件只能通过 ADO 访问数据库中单个基表的数据，它既可以访问一个基表中的所有数据及字段，也可以访问部分记录，即通过在 Filter 属性中设置筛选条件实现。

由于 TADOTable 是从 TDataSet 继承下来的, 所有具有一般数据集的一些通用特点, 同时作为一个 ADP 数据集, 还有自己的一些特点, 添加了一些新的属性、事件及方法, 用于连接到一个 ADO 数据存储、访问底层记录集对象、书签筛选记录、异步提取记录, 以及通过缓存更新来执行批更新、使用磁盘上的文件来保存数据等操作。

ADOTable 组件与其他组件不同的是, 它专门针对数据库中表进行操作。下面是 ADOTable 特有的一些属性方法:

#### (1) MasterSource 属性和 MasterFields 属性

MasterSource 属性是用于建立主从关系的数据源。当前的 ADOTable 中的数据将根据 MasterSource 所指定的数据源变化而变化。在一个数据库汇总, 某些表之间可能存在着互相关联的, 一种常见的情况是两个表之间存在共同的字段。通过这些共同的字段, 可以建立主从关系的关联, 可以通过访问一个表而得到另外一个表的数据。这种主从关系建立之后, 如果用户在主表选中一条记录, 则从表中相应的记录自动获得。

MasterFields 属性指定用于建立主从关系的关联字段。指定的字段是将是主表和从表赖以存在的纽带, 这个字段必须是主从表中共有的字段。

#### (2) ReadOnly 属性

指定 ADOTable 中数据是否处于只读状态。

#### (3) TableDirect 属性

指定是通过表名来访问数据库还是在后台运行 SQL 命令访问数据库。部分的数据提供者不支持通过表名对数据库的访问, 这时的 ADOTable 就只能通过后台运行 SQL 的 SELECT 语句来访问数据库。如果设置为 True, 则 ADOTable 组件在后台运行 SQL 命令访问数据表。为 False, 按表名访问数据库, 缺省值为 False。

#### (4) TableName 属性

指定 ADOTable 要操作的表名。

只有当设置了正确的 Connection 或者 ConnectionString, Active 属性为 True 的时候, 才能设置 TableName 属性, 表名的列表才会在 Object Inspector 中显示。

下面这个例子是演示了从数据库中提取全部的表名, 并一一访问各个表, 查询到每个表的表名和记录数, 并存入另外一个表 Table1:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SL: TStringList;
  index: Integer;
begin
  //创建一个列表实例, 作为GetTableNames的参数用
  SL := TStringList.Create;
  try
    ADOConnection1.GetTableNames(SL, False);
    for index := 0 to (SL.Count - 1) do begin
      Table1.Insert; //插入一条新记录
      //取得表名
      Table1.FieldName('Name').AsString := SL[index];
      if ADOTable1.Active then ADOTable1.Close;
      //准备打开指定的表
    end;
  finally
    SL.Free;
  end;
end;
```

```

        ADOTable1.TableName := SL[index];
        ADOTable1.Open;
        //取得记录总数
        Table1.FieldName('Records').AsInteger := ADOTable1.RecordCount;

        //提交并保存记录
        Table1.Post;
    end;
finally
    SL.Free;
    ADOTable1.Close;
end;
end;

```

#### (5) Append 方法和 AppendRecord(const Values: array of const);

使用 Append 方法可以在数据表中增添一条新的空记录，并将当前的记录指针指向该条空记录。

使用 AppendRecord 方法在增加一条新记录的同时，并为新记录赋值。赋值是通过一个数组作为参数传递进去的，但是必须保证数组的维数和与字段数一致且顺序一致。

#### (6) Post 方法

执行 Post 方法，是将已经修改完毕的记录写入数据库中。通常是每执行完一个修改记录的操作，执行一次 Post 操作，以完成对数据库的更新。

下面的简单例子演示了如何添加一条记录：

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    //添加一条空记录
    Table1.Append;
    //为空记录赋值
    Table1.FieldValues['ALPHANUMERIC'] := Edit1.text;
    Table1.FieldValues['INTEGER'] := StrToInt(Edit2.text);
    //提交并保存记录
    Table1.Post;
end;

```

### 4.2.5 ADOQuery 组件

ADOQuery 组件借助于 SQL 语言的强大功能访问多个数据表，可以实现数据浏览、修改和删除等操作。而且，ADOQuery 组件可以实现参数查询。所谓参数，在使用上可以理解为用户变量，在执行 SQL 之前，就被赋值。运用参数化查询，不需要修改 SQL 语句，给定不同的参数值，就可以获得不同查询结果。通常情况下，使用 ADOQuery 是为了从数据集中查询一部分字段或记录，也可以使用 INSERT、DELETE、UPDATE、ALTER TABLE 等 SQL 命令实现数据库的更新、插入和删除记录的操作。如果数据集只包含一个基表，则可以使用 ADOQuery，也可以使用 TADOTable 数据集。下面讨论一些关于 ADOQuery 的主要属性和方法。

#### (1) SQL 属性

SQL 属性是 TStrings 类型的变量，包含了 ADOQuery 组件要执行的 SQL 命令。它是



ADOQuery 的最为重要的属性之一。在应用程序中，可以调用 Open 方法或 ExecSQL 方法来执行在 SQL 属性中指定的 SQL 语句。在代码编写阶段，可以利用属性编辑器编写；在应用程序执行过程中也可以动态修改，请看下面的例子：

```
with ADOQuery1 do begin
    //重新写入SQL的时候，必须关闭原来的查询
    Close;
    with SQL do begin
        //清楚原来的SQL命令
        Clear;
        Add('SELECT EmpNo, LastName, FirstName, HireDate');
        Add('FROM Employee');
    end;
    //执行SQL命令，获得想要的查询结果
    Open;
end;
```

## (2) Parameters

Parameters 属性中保存了 SQL 属性中的 SQL 命令中执行所需的参数。这些参数可以在程序设计阶段添加。这时可以在 Object Inspector 设定参数的值，并且，参数的数量和类型必须与 SQL 属性中 SQL 语句的参数一致。如图 4-13 所示。

另一种方法，也是常用的方法是在程序执行过程中，根据 SQL 语句的不同可以动态设置参数值。示例如下：

```
with ADOQuery1 do begin
    SQL.Clear;
    SQL.Add('insert into country(name, captial, population)');
    SQL.Add('values(:Name, :Capital, :Population)');
    Parameters.ParamByName('Name').Value := 'Liechtenstein';
    Parameters.ParamByName('Capital').Value := 'Vaduz';
    Parameters.ParamByName('Population').Value := 420000;
    Prepared := true;
    ExecSQL;
end;
```

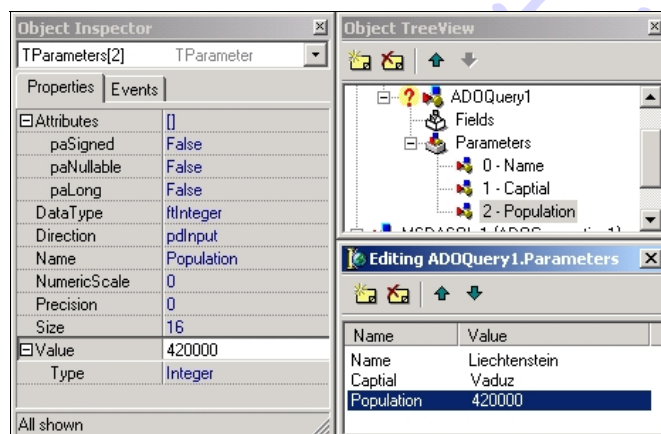


图 4-13 编辑 SQL 语句的参数

### (3) Open 方法和 ExecSQL 方法

这两种方法都用于执行 SQL 属性所指定的 SQL 命令,功能上能基本上类似。但是 Open 方法通常调用 SELECT 语句,是要返回记录集;而 ExecSQL 方法执行 Insert、Update 或 Delete 等命令,不返回记录集。无论是调用 Open 方法还是 ExecSQL 方法,在执行它们之前,都必须调用 Close 方法。

#### 4.2.6 TADOStoredProc 组件

如果一个客户应用程序必须使用数据库中的存储过程,则可以使用 TADOStoredProc 组件。一个存储过程是一组语句,提前建立好保存在数据库服务器上,可以反复被执行,在服务器上完成与数据库有关的任务,并将结果传递给客户。

许多存储过程在进行处理时还需要一系列参数,可以通过 TADOStoredProc 组件的 Parameters 属性来设置各种参数。该数据集与其他 ADO 数据集相似,可完成类似的功能。

1. TADOStoredProc 组件主要的属性 TADOStoredProc 组件与其他 ADO 数据集有相似的属性,经常需要设置的几个属性如下:

- Active 设置为 True 时,可以激活数据集;设置为 False 则关闭数据集。
- Connection 如果应用程序中添加了 TADOConnection 组件,则可以通过选择该组件来建立数据库的连接。
- ConnectionString 如果没有设置 Connection 属性,则可以在该属性中设置一个连接到数据库的字符串。
- DataSource 是另一个数据集对应的数据源,用于为当前的数据集提供一些值。一般情况下,不要设置该属性,更不能将其设置为当前数据集使用的数据源。实际上如果这样做,在设计阶段 Delphi 就会报错。
- Filter 设置筛选记录的条件。
- Filtered 决定是否激活 Filter 中设置的筛选条件。
- Parameters 设置存储过程使用参数的属性。
- ProcedureName 设置数据集使用的存储过程的名字,可以从下拉框中选择。

#### 2. TADOStoredProc 组件主要的方法

- Create 该方法用于建立 TADOStoredProc 组件的一个实例。实际上,当使用 ADO 组件面板上的 TADOStoredProc 组件,在一个表单或数据模块中插入该组件时,就自动建立了一个实例,也就相当于隐含地调用了 Create 方法。所以一般很少直接调用该方法建立 TADOStoredProc 组件的实例。
- Close 用于销毁 TADOStoredProc 组件的一个实例。
- DeleteRecords 用于删除一条或多条记录。
- Edit 设置数据集为编辑状态。
- EnableControls 使数据感知控件重新显示数据。
- DisableControls 使数据感知控件不能显示数据。
- First 导航到数据集的第一条记录。
- Last 导航到数据集的最后一条记录。
- Prior 导航到数据集的前一条记录。

- Next 导航到数据集的下一条记录。
- MoveBy 向前或向后导航多条记录。
- GetFieldData 将一个字段当前的值提取到缓存中。
- IsEmpty 判断一个数据集是否为空。
- Open 打开一个数据集。
- Refresh 重新从数据库中提取数据来更新数据集的数据

### 3. TADOStoredProc 组件的事件

TADOStoredProc 组件与 TADOQuery 组件的事件基本相同,所以可以参考 TADOQuery 组件事件的说明。

#### 4.2.7 TRDSConnection 组件

TRDSConnection 组件用于实现一个 RDS DataSpace 对象。当一个 Recordset (记录集) 对象从一个进程或机器传递到另一个进程或机器上时, RDS DataSpace 对象负责管理数据的汇集。当使用基于 ADO 的业务 (business) 对象 (应用程序服务器) 来建立多层的应用程序时, 应该使用 TRDSConnection 对象。

在建立应用程序时, 使用 TRDSConnection 组件代替 TADOConnection 组件, 建立与一个 TADODataset 数据集的关联关系, 并在 TADODataset 组件的 RDSConnection 属性中选择使用的 TRDSConnection 组件实例。

默认情况下, TRDSConnection 组件与 RDS DataFactory 对象一起使用。如果不需要专门的业务对象, 可以使用 DataFactory 对象。

#### 1. TRDSConnection 组件主要的属性

- AppServer 用于访问应用程序服务器的通信接口。如果需要访问与 Recordsets 对象无关的业务对象的附加属性或方法时, 可以使用该属性。
- ComputerName 指定一个业务对象的来源。如果该属性为空, 则从本地计算机装载业务对象。HTTP、HTTPS 和 DCOM 等协议可以用于建立业务对象。如果使用 HTTP 和 HTTPS 协议建立业务对象, 则 ComputerName 属性是包含确定 IIS Web 服务器的 URL 的一个字符串, 服务器的业务对象实例就在该服务器上建立; 如果使用 DCOM 协议, 则 ComputerName 属性是计算机的名字。
- DataSpaceObject 提供对 RDS DataSpace 对象接口的访问。
- Connected 确定是否已经建立了到远程数据源的连接, 设置为 True 则表示建立了连接。
- InternetTimeout 设置超出规定的请求次数以前使用时间的数量, 以毫秒计算。
- Name 设置 TRDSConnection 组件实例的名字。
- ServerName 指定业务对象, 即需要初始化的业务对象的 ProgID, 默认值是在 RDSServer.DataFactory 属性中, 这是 RDS DataFactory 对象的 ProgID。要了解 DataFactory 对象的更多信息, 可以参考 Microsoft Data Store SDK 帮助文档。

#### 2. TRDSConnection 组件主要的方法

- GetRecordset 用于从一个业务对象中提取一个记录集。与 RDS 连接组件关联的 TADODataset 会自动调用该方法。其语法如下:

```
function GetRecordset(const CommandText: WideString; ConnectionString:
WideString = ''): _Recordset;
```

例如，下面的代码使用该方法建立一个记录集：

```
ADODataset1.CommandText := 'SELECT * FROM Employee';
ADODataset1.RecordSet:=RDSConnection1.GetRecordset (ADODataset1.CommandTe
xt, '');
```

- **Free** 销毁组件的实例并释放其占用的资源。
- **Open** 打开 TRDSConnection 组件的连接。
- **Create** 建立 TRDSConnection 组件的一个实例。当在表单或数据模块中添加一个 TRDSConnection 组件时，就隐含调用了该方法。
- **Destroy** 用于销毁 TRDSConnection 组件的一个实例。在应用程序中一般不要直接调用该方法，而应调用 Free 方法来销毁组件的实例。

### 3. TRDSConnection 组件主要的事件

- **AfterConnect** 该事件在建立 TRDSConnection 组件的连接后触发。
- **AfterDisconnect** 该事件在断开 TRDSConnection 组件的连接后触发。
- **BeforeConnect** 该事件在建立 TRDSConnection 组件的连接前触发。
- **BeforeDisconnect** 该事件在断开 TRDSConnection 组件的连接前触发。
- **OnLogin** 该事件在正确进行了用户注册并打开了到服务器的通信通道后触发。

## 4.3 ADO 应用实例

下面通过几个实例说明如何应用 ADO 面板的常用组件。

### 4.3.1 使用 ADODataset 实现数据查询

建立该应用程序的步骤如下：

1. 建立应用程序 使用 File/New/Application 新建一个应用程序。
2. 添加组件 在表单 Form1 中添加下列组件：
  - (1) 通过 Data Access 面板的 TDataSource 组件添加一个数据源 DataSource1。
  - (2) 通过 ADO 面板的 TADODataset 组件添加一个数据集 ADODataset1。
  - (3) 通过 Data Control 面板的 TDBGrid 组件添加一个数据表格 DBGrid1。
  - (4) 通过 Data Control 面板的 TDBNavigator 组件添加一个数据导航条 DBNavigator1。
3. 设置各个组件的属性

(1) 通过 ADODataset1 组件的 ConnectionString 属性建立数据库的连接，所以不需要使用 ADO 面板的 TADOConnection 组件。

单击 ConnectionString 属性右边的按钮打开连接字符串设置面板并单击 Build 按钮，然后在打开的“数据链接属性”面板的“提供者”标签页选择 Microsoft OLE DB Prvider for ODBC Drivers，在“连接”标签页的“使用数据源名称”中选择 dbdemos（假设已经在控制面板的 ODBC 管理器中建立了该数据源，它使用的是 Delphi6 提供的 Access 数据库 dbdemo.mdb）。再单击“测试连接”按钮，检验是否能够正确连接到数据库。最后确认后逐级关闭打开的面板。则 ConnectionString 属性会显示出一个连接到数据库的字符串。

(2) 各个组件的属性 各个组件属性的设置值见表 4-9。

表 4-9 各个组件属性的设置值

组件名称	属 性	设 置 值	说 明
ADODataSet1	ConnectionString	Provider=MSDASQL.1;Persist Security Info=False;DataSource=dbdemos	建立数据库连接的字符串
	CommandType	cmdTable	通过基表访问数据
	CommandText	employee	具体使用的基表名字
	Filter	LastName like 'b%' and EmpNo>100	设置数据集的筛选条件
	Filtered	True	使 Filter 中设置的筛选条件起作用
	Active	True	激活数据集
DataSource1	DataSet	ADODataSet1	设置与数据源连接的数据集
DBGrid1	DataSource	DataSource1	使用的数据源
DBNavigator1	DataSource	DataSource1	使用的数据源

最后设计完成的界面如图 4-14 所示。现在运行程序，结果与该设计界面类似。

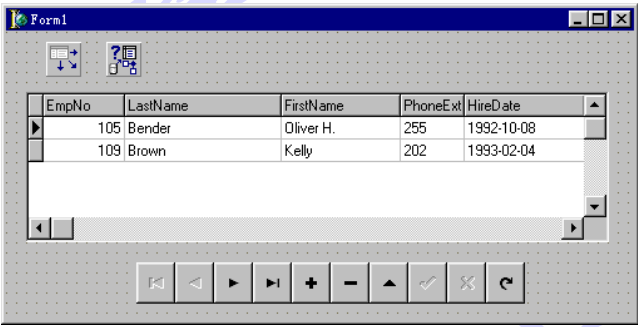


图 4-14 通过 TADODDataSet 组件建立的数据库应用程序界面

4. 修改程序 下面我们在应用程序运行时修改 ADODDataSet1 组件中 Filter 属性的值，使查询的结果改变。方法如下：

(1) 设置组件属性 在表单 Form1 顶部添加一个标签 Label1、一个文本编辑框 Edit1 和一个按钮 Button1，这三个组件属性的设置值见表 4-10。

表 4-10 新添加三个组件属性的设置值

组件名称	属 性	设 置 值	说 明
Label1	Caption	输入 LastName 字段开头字母:	文本标题
	Font	楷体/14 号	设置标题字体
Edit1	Text		将原来的文本 Edit1 删除
Button1	Font	楷体/14 号	按钮标题的字体
	Caption	重新查询	按钮标题

(2) 添加按钮执行的代码 双击按钮 Button1，打开代码编辑器并添加代码，其完整

的代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ADODataSet1.Filtered:=False;    //使Filter属性失效
    ADODataSet1.Filter:='LastName like '+''+Trim(Edit1.Text)+#37+''';
                                //使用 Edit1中输入内容重新设置Filter属性
    ADODataSet1.Filtered:=True;    //激活Filter属性
end;
```

由于数据集的 Filter 属性值是一个字符串,所以给该属性赋的值要使用字符串。第一个字符串为'LastName like ',注意 like 后面要保留一个空格,否则与后面的字符串就连接到了一起。""表示一个单引号;Trim(Edit1.Text)表示删除 Edit1 输入内容前后的空格;#37 代表%,因为%的 ASCII 字符值是 37,不能在代码中直接使用%。Delphi 中表示一些特殊字符时,要在该字符 ASCII 前面添加一个#表示该字符。

比如,如果 Edit1 中输入的字符是 a,则第二行代码执行的结果相当于在 ADODataSet1 数据集的 Filter 属性中设置如下值:

```
LastName like 'a%'
```

**注意:**数据集的 Filter 属性中字段名及条件不区分大小写。

(3) 现在运行程序,并在 Edit1 中输入如下字母 g,则单击按钮 Button1 后显示的结果如图 4-15 所示。



图 4-15 修改 Filter 属性值后的查询结果

#### 4.3.2 使用 TADOQuery 实现多表查询

建立该应用程序的步骤如下:

1. 建立应用程序 使用 File/New/Application 新建立一个应用程序。

2. 添加组件 在表单 Form1 中添加下列组件:

(1) 分别通过 ADO 面板的 TADOConnection 和 TADOQuery 组件,在表单 Form1 中分别添加一个数据库连接组件 ADOConnection1 和数据集组件 ADOQuery1。

(2) 通过 Data Access 面板的 TDataSource 组件添加一个数据源 DataSource1。

(3) 通过 Data Control 面板的 TDBGrid 组件添加一个数据表格 DBGrid1。

(4) 通过 Data Control 面板的 TDBNavigator 组件添加一个数据导航条 DBNavigator1。

3. 设置各个组件的属性

(1) 通过 ADOConnection1 组件的 ConnectionString 属性建立数据库的连接,其建立方法与实例 1 中 ADODataSet1 的 ConnectionString 属性建立数据库连接的方法及结果完全



一样。  
(2) 各个组件的属性 各个组件属性的设置值见表 4-11。

表 4-11 各个组件属性的设置值

组件名称	属 性	设 置 值	说 明
ADOConnection1	ConnectionString	Provider=MSDASQL.1;Persist Security Info=False;Data Source=dbdemos	建立数据库连接的字符串
ADOQuery1	Connection	ADOConnection1	
	SQL	Select Items.OrderNo,Items.ItemNo,Items.Qty, Orders.CustNo,Orders.SaleDate from Items,Orders where Items.OrderNo=Orders.OrderNo and Items.OrderNo<1005;	
	Active	True	激活数据集
DataSource1	DataSet	ADOQuery1	设置与数据源连接的数据集
DBGrid1	DataSource	DataSource1	使用的数据源
DBNavigator1	DataSource	DataSource1	使用的数据源

ADOQuery1 的 SQL 语句用于查询 Items 和 Orders 两个关联基表的数据，记录的对应关系是两个基表都有的 OrderNo 字段对应相等。字段前面都添加了基表的名字，用于显示该字段的归属关系。

最后设计完成的界面如图 4-16 所示，其中前 3 个字段属于 Items 基表，后面的两个字段属于 Orders 基表。现在运行程序，结果与该设计界面类似。

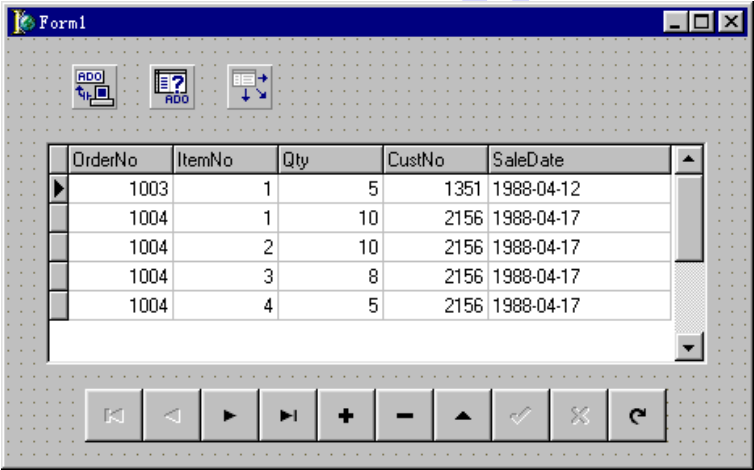


图 4-16 通过 TADOQuery 组件查询两个关联基表的数据

4. 与实例 1 的比较 实例 1 中没有使用 TADOConnection 组件建立数据库的连接，而是直接通过设置数据集的 ConnectionString 属性建立数据库的连接；实例 2 使用 TADOConnection 组件建立数据库的连接建立数据库的连接，但是也是通过设置 TADOConnection 组件的 ConnectionString 属性建立数据库的连接。可见，两个实例中建立数据库连接方法的本质是一样的。所以为了简便，一般应用程序中不需要使用

TADOConnection 组件建立数据库的连接。

**4.3.3 使用 TADOCommand 组件**

TADOCommand 组件可以通过 SELECT 语句返回一个结果集，此时该组件相当于 TADOTable、TADOQuery 等数据集组件实现的功能；它也可以执行没有返回结果集的命令，此时需要使用 INSERT、DELETE 或 UPDATE 等 SQL 语句。该实例提供的功能是通用的，即 TADOCommand 组件根据使用的命令不同，可以返回结果集，也可以不返回结果集。

该应用程序的实现步骤如下：

1. 建立应用程序 使用 File|New|Application 新建一个应用程序。
2. 添加组件 在表单 Form1 中添加下列组件：
  - (1) 通过 ADO 面板的组件，在表单 Form1 中分别添加一个数据库连接组件 ADOConnection1、一个 ADO 命令组件 ADOComman1、一个数据集组件 ADODDataSet。
  - (2) 通过 Data Access 面板的 TDataSource 组件添加一个数据源 DataSource1。
  - (3) 通过 Standard 组件面板，在 Form1 中添加一个面板 Panel1、一个标签 Label1、一个多行文本输入组件 Memo1 和一个按钮组件 Button1。
  - (4) 通过 Data Control 组件面板，添加一个数据表格 DBGrid1 和一个数据导航条 DBNavigator1。
3. 设置各个组件的属性  
各个组件属性的设置值见表 4-12。

表 4-12 各个组件属性的设置值

组件名称	属性	设置值	说明
ADOConnection1	ConnectionString	Provider=MSDASQL.1;Persist Security Info=False;Data Source=dbdemos	建立数据库连接的字符串
ADOCommand1	Connection	ADOConnection1	
	CommandType	cmdText	
ADODDataSet1	Connection	ADOConnection1	
	CommandType	cmdText	
DataSource1	DataSet	ADODDataSet1	设置与数据源连接的数据集
DBGrid1	DataSource	DataSource1	使用的数据源
DBNavigator1	DataSource	DataSource1	使用的数据源
Panel1	Caption		删除面板标题
	BevelOuter	bvLowered	使面板下凹
Label1	Caption	输入 SQL 命令:	
Memo1	Lines		删除该组件中的文本
Button1	Caption	执行命令	

以上所有组件的属性设置完毕后，表单 Form1 的设计界面如图 4-17 所示。

**4. 添加程序的运行代码**

应用程序运行后，可以在 Memo1 中输入各种 SQL 命令，然后单击按钮 Button1，则应该执行这些命令。由于插入的命令中，使用 SELECT 语句的会返回一个结果集，而其他 SQL

语句则不会返回结果集，所以通过检查用户输入的 SQL 命令来决定执行哪一部分代码。

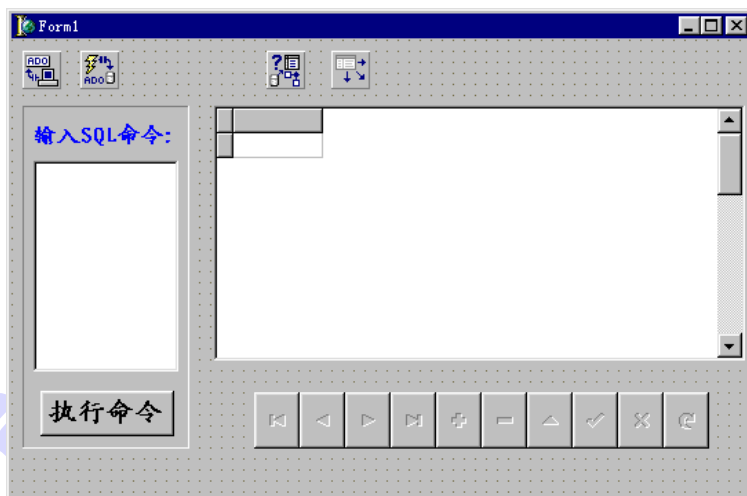


图 4-17 设置完组件属性后的 Form1 设计界面

该按钮事件处理器的完整代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
Var
    str1,str2:String;
begin
    str1:= Trim(Memo1.Text);           //删除SQL命令两边多余的空格
    str2:= copy(str1,0,6);             //拷贝输入SQL命令中前6个字符
    ADOCommand1.CommandText:= str1;    // ADOCommand1执行的命令

    If str2='select' then               //检查用户输入的是不是SELECT语句
    Begin                               //有结果集返回时执行的代码
        ADODataset1.Recordset:=ADOCommand1.Execute;
        ADODataset1.Open;
    End
    Else                               //无结果集返回时执行的代码
    Begin
        ADODataset1.Close;             //在运行ADOCommand1前要关闭数据集
        ADOCommand1.Execute;           //运行INSERT、DELETE、UPDATE等命令

        ADODataset1.CommandText:='Select * from Country'; //重新显示数据集数
据
        ADODataset1.Open;
    End;
End;
```

## 5. 运行程序

现在可以运行程序，并在 Memo 中输入各种 SQL 语句。比如，输入 SELECT 语句时的显示如图 4-18 所示。

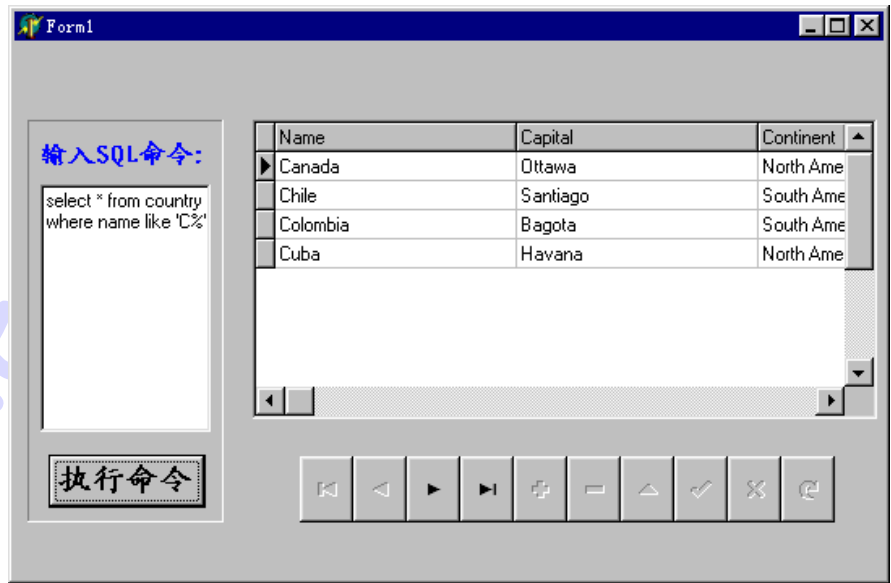


图 4-18 执行 SELECT 命令的结果

## 第 5 章 开发 InterBase 数据库应用程序

InterBase 数据库是 Borland 公司开发的数据库，在 Borland 产品中获得了广泛的应用。该数据库既可以安装在本地计算机上，也可以安装在远程服务器上。并且该数据库是 Delphi 或 Kylix 开发的跨平台应用程序可以使用的数据库之一（还可以使用 Oracle、MySQL 和 DB2），这两个开发工具中提供了许多功能强大的 InterBase 数据库开发组件，所以如果选择 Delphi 或 Kylix 作为开发工具，使用 InterBase 作为应用程序使用的数据库则是一个不错的选择。它们有结合非常紧密的特点。并且在 Delphi 的安装盘中提供了 InterBase 数据库服务器和客户端软件，可以很方便地构筑 InterBase 数据库的开发环境。

### 5.1 InterBase 面板

InterBase 面板提供的组件可以直接连接到 InterBase 数据库并访问该数据库中的数据，并且速度较快，设置也比较简单，而不需要 ADO 或 BDE 等。该面板的许多组件都可以在 BDE 或 ADO 面板找到实现相似功能的组件，这些类似组件的属性、方法及事件等也比较相似，比如 IBTable 与 TTable、TADOTable，IBQuery 与 TQuery、TADOQuery 等比较相似。

InterBase 面板包括的组件及其功能如下：

- IBTable 表示一个 InterBase 基表或一个 InterBase 视图中所有或部分数据。
- IBQuery 使用 SQL 语句从一个或多个 InterBase 基表中提取数据。如果要从本地的 InterBase 数据库移植到远程的 InterBase 数据库，则该数据集比其他数据集有更多的伸缩性。
- IBStoredProc 该组件用于运行一个 InterBase 存储过程，它不能表示一个结果集，如果一个数据库存储过程有结果集返回，则应该使用 IBQuery 或 IBDataSet 组件。
- IBDatabase 用来表示一个 InterBase 数据库连接。使用该组件可以管理事务或为远程数据库提供连接参数。
- IBTransaction 对一个或多个数据库提供不连续的事务控制，IBDatabase 组件使用 IBTransaction 来表示一个事务。
- IBUpdateSQL 该组件用于使用缓冲的更新来支持只读的查询。可以使用 IBUpdateSQL 组件和封装的查询来提供一种更新数据集的方法，从而可以将更新提交到一个只读的数据集中。
- IBDataSet 用于表示来自 SQL SELECT 命令的结果集，IBDataSet 组件可用于指定单独的 SQL 命令用于插入、删除或更新记录。
- IBSQL 该组件可以使用最小的开销来运行一个 InterBase SQL 语句，该组件没有到数据感知组件的标准接口，并且是单方向的。
- IBDatabaseInfo 返回一个连接数据库的信息，比如在线磁盘结构（Online Disk Structure, ODS）的版本、分配的缓存区的数量、读写数据库页面的数量及写在前面的日志信息等。

- **IBSQLMonitor** 监视传递到 InterBase 服务器的动态的 SQL 语句，它引入了一个事件为 OnSQL，该事件用于接收基于服务器的每个动态 SQL 语句的文本。
- **IBEvents** 使一个应用程序注册表有兴趣异步处理被 InterBase 服务器传递的事件。
- **IBExtract** 该组件用于提取元数据信息，比如 InterBase 基表、视图、角色或索引的一个列表。
- **IBClientDataSet** 使用 InterBase Express 而不是使用一个外部的提供者及客户数据集来提取数据。

**注意：**要使用 InterBase 面板的组件，就必须先安装 InterBase 数据库，否则将无法使用该面板的组件。

### 5.1.1 TIBTable 组件

该组件可以直接访问 InterBase 数据库一个基表或一个视图中的每个记录或字段，这些记录和字段可以是一个基表或视图的全部数据，也可以是部分数据。

#### 1. TIBTable 组件主要的属性

- **Active** 决定是否激活当前的数据集。
- **BufferChunks** 用于指定数据集缓存块的大小，默认是 1000 字节。注意，当 Unidirectional 属性为 True 时，则该属性的设置没有意义，因为此时不需要缓存。
- **CachedUpdates** 决定是否启用缓存。
- **Database** 指定用于连接到 InterBase 数据库的 TIBDatabase 组件。
- **FieldDefs** 可以通过该属性设置基表中各个字段的属性。
- **Filter** 设置基表的筛选条件。
- **Filtered** 设置是否激活筛选条件。
- **IndexFieldNames** 列出作为索引的字段。
- **ReadOnly** 决定该数据集是否是只读的。
- **TableName** 决定使用的基表或视图的名字。
- **TableTypes** 决定基表的类型，ttSystem 表示使用基表，ttView 表示使用视图。
- **Transaction** 选择一个 TIBTransaction 组件用于事务管理。
- **Bof** 数据集第一条记录。
- **Eof** 数据集最后一条记录。

#### 2. TIBTable 组件主要的方法

- **AddIndex** 用于为基表建立一个新的索引。其语法如下：  

```
AddIndex(const Name, Fields: string; Options: TIndexOptions const DescFields: string = '');
```
- **DeleteIndex** 用于删除一个索引。其语法如下：  

```
procedure DeleteIndex(const Name: string);
```
- **DeleteTable** 删除一个已经存在的基表，无参数。
- **EmptyTable** 删除基表中所有的数据，无参数。
- **Close** 关闭当前的数据集。



- Edit 使当前数据集进入编辑状态。
- Open 打开当前的数据集。

### 3. TIBTable 组件主要的事件

After、Before 和 On 开头的事件分别发生在某种情形之前、之后或正在发生。另外，还有两个该数据集特有的事件如下：

- DatabaseFree 当销毁了 Database 属性中设置的数据库组件并且释放其占用的内存时触发该事件。
- TransactionFree 当销毁了 Transaction 属性中设置的数据库事务组件并且释放其占用的内存时触发该事件。

#### 5.1.2 TIBQuery 组件

##### 1. TIBQuery 组件主要的属性

- Active 决定是否激活该数据集。
- Database 设置使用连接到数据库的 TIBDatabase 组件。
- ParamCheck 指定如果运行时 SQL 属性改变，是否为一个查询重新建立参数列表。
- Params 用于设置一个查询的 SQL 语句中使用的参数。
- SQL 设置一个查询运行的 SQL 语句。
- Transaction 确定查询执行的事务。
- UpdateObject 设置一个 TIBUpdateSQL 更新对象，会自动应用缓冲区的更新。

##### 2. TIBQuery 组件主要的方法

- BatchInput 在 SQL 语句中运行一个参数化的查询，用于输入可参考的输入对象。
- BatchOutput 向参考的 OutputObject 对象输出 SQL 语句中选择的查询。
- ExecSQL 执行查询的 SQL 语句。
- GetDetailLinkFields 该方法会建立两个字段列表，一个用于显示主表的字段，一个用于显示从属表的字段。
- ParamByName 按照特定的参数名称访问参数信息。
- Prepare 先向服务器发送一个查询来优化以前的运行。
- UnPrepare 释放前面分配给预备查询语句的资源。

##### 3. TIBQuery 组件主要的事件

TIBQuery 组件的事件与 TIBTable 组件的事件基本相同。可参考 TIBTable 组件的事件。

#### 5.1.3 IBStoredProc 组件

当一个客户应用程序必须使用数据库服务器上的一个存储过程时，可以使用 TIBStoredProc 对象。一个存储过程是一系列语句，作为数据库服务器的元数据保存在数据库中，可以被重复执行，完成一些与数据库相关的任务，并将结果传递到客户端。

在执行一个存储过程中可能需要多个参数，可以通过 TIBStoredProc 对象的 Params 属性提供这些参数。TIBStoredProc 组件只用于没有返回结果的 InterBase 可执行存储过程；如果要执行有返回结果的 InterBase 存储过程，应该使用 TIBQuery 或 TIBDataSet 组件，而一定不能使用 Open 或 Active 属性来激活一个 TIBStoredProc 组件，应该用 ExecProc 方法代

替。

#### 1. TIBStoredProc 组件主要的属性

- **AutoCalcFields** 决定是否触发 OnCalcFields 事件及何时计算查询字段的值。
- **Database** 确定连接到数据库的 TIBDatabase 组件。
- **Filtered** 决定是否激活数据集的筛选器。
- **Name** 设置存储过程组件实例的名字。
- **ObjectView** 决定在 Fields 属性中, 哪些字段按照多层次的方式存储, 哪些字段放在同一个层次上存储。
- **Params** 设置存储过程使用的参数。
- **StoredProcName** 选择一个已经在数据库服务器上建立的存储过程的名字。
- **Transaction** 确定执行查询时使用的事务。

#### 2. TIBStoredProc 组件主要的方法

- **CopyParams** 将一个存储过程的参数拷贝到另一个存储过程中。比如:  
`IBStoredProc1.CopyParams (IBStoredProc2.Params);`
- **ExecProc** 在服务器上运行当前的存储过程。在使用该方法以前, 应该在 Params 属性中提供任何数据参数并调用 Prepare 方法来绑定这些参数。如果一个存储过程有输出参数, 则执行 ExecProc 方法后会将这些参数保存到 Params 属性中。

例如, 执行一个存储过程的代码如下:

```
IBStoredProc1.Prepare;
try
    IBStoredProc1.Params[0].AsString := Edit1.Text;
    IBStoredProc1.ExecProc;
    Edit2.Text := IBStoredProc1.ParamByName('FinalValue').AsString;
    // FinalValue为参数的名字
finally
    IBStoredProc1.UnPrepare;
end;
```

- **ParamByName** 为特定的参数名称分配参数信息。比如:

```
StoredProc1.ParamByName('Param1').AsString := 'LiMing';
```

- **Prepare** 准备一个存储过程用于运行, 此时会为该存储过程分配资源。
- **UnPrepare** 释放使用 Prepare 方法时分配给存储过程的资源

#### 3. TIBStoredProc 组件主要的事件

与 TIBTable 组件的事件基本相同。

### 5.1.4 TIBDatabase 组件

TIBDatabase 组件用于建立到 InterBase 数据库的连接, 所有的 InterBase 数据集组件和 TIBSQL 组件都要通过 TIBDatabase 组件来访问数据库。

#### 1. TIBDatabase 组件主要的属性

- **AllowStreamedConnected** 确定是否可以在设计时设置 Connected 属性。如果该属性使用默认值 True 时, 只要 Connected 也设置为 True 就可以在运行时自动连接到数据库; 如果该属性设置为 False, 则只能在运行时将 Connected 设置为 True 来连

接到数据库，或者调用 Open 方法。

- **Connected** 确定是否激活到数据库的连接。
- **DatabaseName** 指定要连接的 Interbase 数据库的名字。如果该数据库在本地，则可以使用该数据库文件名的完成路径设置该属性；如果数据库在远程服务器上，则对应使用的不同网络类型而使用如下三种不同的设置方法：
  - ◇ TCP/IP 协议 <服务器名字>:<文件名>，比如 myServer:c:\data\employee.gdb
  - ◇ NetBEUI 协议 \\<服务器名字>\<文件名>，比如\\myServer\c:\data\employee.gdb
  - ◇ SPX 协议 <服务器名字>@<文件名>，比如 myServer@c:\data\employee.gdb
- **DefaultTransaction** 设置或返回默认的数据库事务。可以选择一个已经建立的 TIBTransaction 对象的名字。
- **IdleTimer** 指定断开一个闲置的连接以前需要等待的时间。
- **LoginPrompt** 如果该属性使用默认的设置 True，则程序运行时会显示一个标准注册窗口用于注册用户名及口令。设置为 False 则屏蔽该窗口。
- **Name** 设置该组件的名字。
- **Params** 设置传递到 InterBase 服务器的数据库参数。
- **SQLDialect** 设置或返回客户使用的特定语言。
- **TraceFlags** 设置应用程序运行时 SQL 监视器使用的跟踪操作方式，各个设置值的意义见表 5-1。

表 5-1 各种操作方式及意义

操作方式	意义
tfQPrepare	监视 Prepare 语句
tfQExecute	监视 ExecSQL 语句
tfError	监视服务器错误信息。这些信息可能包括一个错误代码
tfStmt	监视所有的 SQL 语句
tfConnect	监视数据库连接和断开连接的操作，包括连接句柄的分配和释放连接句柄
tfTransact	监视一些事务语句，比如 StartTransaction、Commit 和 Rollback 语句
tfBlob	监视 blob 数据类型的操作
tfMisc	监视没有被其他标志选项覆盖的语句
tfQFetch	监视 Fetch 语句
tfService	监视服务

## 2. TIBDatabase 组件主要的方法

- **Close** 关闭数据库的连接。
- **Free** 销毁该数据库连接并释放其使用的内存。
- **Open** 打开该数据库连接。

## 3. TIBDatabase 组件主要的事件

- **AfterConnect** 在建立数据库连接之后触发该事件。
- **AfterDisconnect** 在断开数据库连接之后触发该事件。
- **BeforeConnect** 在建立数据库连接之前触发该事件。

- BeforeDisconnect 在断开数据库连接之前触发该事件。
- OnDialectDowngradeWarning 在客户连接的 SQL 特定语言降级时触发。
- OnIdleTimer 在数据库连接空闲后触发该事件。
- OnLogin 在一个应用程序成功注册并连接到数据库后触发该事件。

#### 5.1.5 TIBTransaction 组件

InterBase 组件面板上的所有数据集及 TIBSQL 组件都需要使用一个 TIBTransaction 组件和一个 TIBDatabase 组件来实现对数据库的访问，这与以前使用的其他类型的数据集是不同的。

**注意：**在将一个 InterBaseExpress 数据集连接到一个客户数据集的应用程序中，每个查询必须使用它自己的事务，也就是说，每个 TIBQuery 组件都要使用一个单独的 TIBTransaction 组件。

##### 1. TIBTransaction 组件主要的属性

- Active 决定是否激活当前的事务。
- AutoStopAction 决定自动结束事务后采取何种动作。可设置的动作及其作用见表 5-2。

表 5-2 AutoStopAction 属性可设置的属性值及意义

可设置的动作	意义
saNone	没有隐含停止的事务
saRollback	当隐含停止事务时回滚并关闭事务
saCommit	当隐含停止事务时提交并关闭事务
saRollbackRetaining	当最后数据集关闭时没有停止事务，但是与当前事务相关的数据更新、插入及删除被回滚，只有 InterBase 6.0 以上的版本支持该事务
saCommitRetaining	当最后数据集关闭时没有停止事务，但是与当前事务相关的数据更新、插入及删除被提交

- DefaultAction 设置事务默认的动作。各个动作的意义见表 5-3。

表 5-3 DefaultAction 各个动作的意义

可设置的动作	意义
taRollback	回滚该事务
taCommit	提交该事务
taRollbackRetaining	回滚该事务，但是保持当前的事务内容，只有 InterBase 6 以上版本支持该特点
taCommitRetaining	提交该事务，但是保持当前的事务内容，

- DefaultDatabase 设置该事务使用的默认数据库。
- IdleTimer 指定自动提交或回滚后，该事务需要等待的时间。
- Name 该事务的名字。
- Params 设置或获得该事务使用的参数。

##### 2. TIBTransaction 组件主要的方法

- AddDatabase 将一个数据库与该事务关联，其语法如下：

```
function AddDatabase(db: TIBDatabase): Integer;
```

- **CheckInTransaction** 检查事务是否为激活的，及在事务的数据库列表中是否有数据库列出。
  - **Commit** 提交一个事务，使用将与之相关的数据对数据库进行更新、插入或删除等。
  - **CommitRetaining** 提交当前活动的事务，并在提交后保留事务的内容。
  - **RemoveDatabase** 取消当前事务与一个数据库的关联。
  - **RemoveDatabases** 取消所有与当前事务关联的数据库。
  - **Rollback** 取消当前事务中所有的更新、插入及删除操作，并终止该事务。
  - **RollbackRetaining** 取消当前事务中所有的更新、插入及删除操作，并保持该事务的内容。
  - **StartTransaction** 启动一个新的事务。
3. **TIBTransaction** 组件主要的事件
- **OnIdleTimer** 在一个事务已经闲置时触发该事件。

#### 5.1.6 TIBUpdateSQL 组件

该对象提供 SQL 语句用于更新 TIBQuery 组件表示的只读数据集，此时必须将缓存更新激活。可以根据设计或环境的要求，如果按照设计将一个数据集设置为只读，则应用程序不能提供一个用户界面来更新数据，但是可以在后面初始化一个程序化的模式。

TIBUpdateSQL 组件能够使一个开发者提供 UPDATE、DELETE 和 REFRESH 语句在其他的只读数据集上执行单独的更新查询，这样，单独的更新查询对于终端用户是透明的。

在实际应用中，TIBUpdateSQL 一般放置在一个数据模块或一个表单中，并且通过 TIBQuery 组件的 UpdateObject 属性连接到该组件。如果该属性指向一个有效的 TIBUpdateSQL 对象，则在应用更新的缓存时，属于更新对象的 SQL 语句会自动应用。

##### 1. TIBUpdateSQL 组件主要的属性

- **DeleteSQL** 指定一个 SQL DELETE 语句，当应用缓冲区更新时会删除一条记录。
- **InsertSQL** 指定一个 SQL INSERT 语句，当应用缓冲区更新时会插入一条记录。
- **ModifySQL** 指定一个 SQL UPDATE 语句，当应用缓冲区更新时会更新一条记录。
- **Name** 设置 TIBUpdateSQL 组件的名字。
- **RefreshSQL** 用于访问封装了 RefreshSQL 语句的 SQL 对象。
- **SQL** 当应用缓存中的更新时返回特定的 SQL 语句。其语法如下：

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

其中 UpdateKind 可取的三个值的意义如下：ukModify 返回使用 UPDATE 的 SQL 语句，ukInsert 返回 INSERT SQL 语句，ukDelete 返回 DELETE SQL 语句。

##### 2. TIBUpdateSQL 组件主要的方法

- **Apply** 设置特定 SQL 语句类型的参数，并执行结果语句。其语法如下：

```
procedure Apply(UpdateKind: TUpdateKind);
```

- **ExecSQL** 运行特定类型的 SQL 语句来执行其他只读数据集的更新，此时必须使缓存可用。

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```



- **SetParams** 在 SQL 语句执行前来绑定需要的参数。其语法如下：

```
procedure SetParams(UpdateKind: TUpdateKind);
```

### 5.1.7 TIBDataSet 组件

该组件可用于运行 InterBase SQL 语句，每个 TIBDataSet 组件都有用于 SQL 语句提取数据的属性，一般为 SQL SELECT 语句，这些语句可以修改记录、插入记录、删除记录及刷新数据集等。

TIBDataSet 组件可以缓存结果集，并且可以上下滚动数据，它可以与各种数据感知控件结合紧密。

#### 1. TIBDataSet 组件主要的属性

- **Active** 确定是否激活该数据集。
- **Database** 确定连接到数据库使用的 TIBDatabase 组件。
- **DeleteSQL** 用于输入 SQL DELETE 语句用于删除数据集的一个或多个行。
- **ForcedRefresh** 设置是否在每次将数据提交到数据库中后都要刷新数据。
- **InsertSQL** 设置一个 SQL INSERT 语句，用于向数据集中插入多行数据。
- **ModifySQL** 设置一个 SQL UPDATE 语句，用于修改数据集中多行数据。
- **RefreshSQL** 用于直接访问封装在刷新的 SQL 语句中的 SQL 对象。
- **SelectSQL** 设置一个 SQL UPDATE 语句，用于访问数据集中多行数据。
- **UpdateObject** 可以选择一个 TIBUpdateSQL 对象，用于更新一个只读的结果集。

#### 2. TIBDataSet 组件主要的方法

- **BatchInput** 用于执行参数化的 SQL 查询，参考输入对象的输入。
- **BatchOutput** 向参考输出对象输出 SQL 中的选择查询。
- **ExecSQL** 运行 SelectSQL 属性指定的 SQL 语句，该语句不是一个 SELECT 语句。
- **ParamByName** 返回提取数据集数据的 SELECT 查询的特定参数。
- **Prepare** 准备数据集中要执行的所有查询。
- **UnPrepare** 重置数据集内部查询的状态。

#### 3. TIBDataSet 组件主要的事件

- **After** 开头的事件 这类事件在某种情形出现以后触发。比如 **AfterDelete** 在删除一条记录后触发；**AfterEdit** 在应用程序开始编辑一条记录后触发；**AfterOpen** 在应用程序打开一个数据集而没有进行数据访问后触发。
- **Before** 开头的事件 这类事件在某种情形出现前触发。比如 **BeforeCancel** 在应用程序运行一个查询来取消对活动记录的改变前触发；**BeforeInsert** 在应用程序进入插入模式前触发；**BeforePost** 在应用程序将活动记录的改变提交到数据库前触发。
- **On** 开头的事件 表示某种情形正在执行时触发。比如 **OnEditError** 表示应用程序试图修改或插入一条记录而提交一个异常时触发；**OnUpdateError** 在将缓存的更新应用到数据库时产生了一个异常时触发；**OnUpdateRecord** 表示将缓存的更新应用到一条记录时触发。
- **DatabaseFree** 当与该数据集关联的数据被销毁并释放占用的内存时触发该事件。
- **TransactionFree** 当包括该数据集的查询事务被销毁并释放占用的内存时触发该



事件。

#### 5.1.8 TIBSQL 组件

TIBSQL 组件使用最小的系统开销来执行一个 InterBase SQL 语句，它没有到数据感知控件的标准接口，并且是单方向的。

##### 1. TIBSQL 组件主要的属性

- Database 设置或返回与该查询关联的一个 TIBDatabase 对象。
- GoToFirstRecordOnExecute 如果该属性为 True，则会导航到数据集的第一条记录并打开它。它主要用于 TIBDataSet 中。
- Name 指定该组件实例的名字。
- ParamCheck 指定在运行时改变了 SQL 属性时，一个 SQL 查询的参数列表是否需要重新产生。
- SQL 设置需要运行的 SQL 查询。
- Transaction 设置查询使用的 TIBTransaction 对象实例。

##### 2. TIBSQL 组件主要的方法

- Call 返回基于错误代码的错误信息。其语法如下：

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean):
ISC_STATUS;
```

- CheckClosed 如果没有关闭查询，则提交一个异常。
- CheckOpen 如果查询没有打开，则提交一个异常。
- CheckValidStatement 如果查询没有一个有效的语句，则提交一个异常。
- Close 关闭该查询。
- ExecQuery 执行一个 SQL 查询，没有参数。
- FreeHandle 释放与该查询关联的 InterBase 资源。
- GetUniqueRelationName 获得该查询唯一有关的名字。
- ParamByName 基于特定的参数名字访问参数的信息。
- Prepare 准备要执行的一个查询。

##### 3. TIBSQL 组件主要的事件

- OnSQLChanging 当 SQL 查询正要修改时触发该事件。

#### 5.1.9 TIBDatabaseInfo 组件

该组件可以提取连接数据库的信息，比如附加装置使用的在线磁盘结构的版本(ODS)、分配数据库缓存的数量、读写的数据库页面或写在前面的日志信息等。

##### 1. TIBDatabaseInfo 组件主要的属性

- Database 设置或返回该组件使用的 TIBDatabase 对象实例。
- Name 设置该组件的名字。
- Tag 该属性没有实际意义，可在程序运行时保存一些数值。

##### 2. TIBDatabaseInfo 组件主要的方法

- Call 返回基于错误代码的一条错误消息，其语法如下：

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean):
ISC_STATUS;
```

- **GetLongDatabaseInfo** 返回一个指定的长整数来描述相关数据库的一些方面。其语法如下：

```
function GetLongDatabaseInfo(DatabaseInfoCommand: Integer): Long;
```

#### 5.1.10 TIBSQLMonitor 组件

该组件用于监视出现在 InterBase 应用程序中的动态 SQL 语句，为了使该 SQL 监视器接收到来自各个数据库连接的状态信息，要激活各个 TIBDatabase 组件的 TraceFlags 属性。

##### 1. TIBSQLMonitor 组件主要的属性

- **Enabled** 如果该属性设置为 True，则会激活该 SQL 监视组件。
- **Name** 设置该组件的名字。
- **TraceFlags** 设置应用程序运行时 SQL 监视器使用的跟踪操作方式，该属性各个值的意义见表 5-1。

##### 2. TIBSQLMonitor 组件主要的方法

- **Free** 销毁该对象，并释放其占用的资源

##### 3. TIBSQLMonitor 组件主要的事件

- **OnSQL** 该事件用于在 InterBase 应用程序中编写一个事件处理器来报告动态的 SQL 活动。

#### 5.1.11 TIBEvents 组件

TIBEvents 组件允许应用程序注册或异步处理 InterBase 服务器传递来的事件，InterBase 事件允许应用程序响应其他应用程序修改的动作和数据库，并同时运行应用程序，不需要在一个规则的基础上处理数据库的信息或者直接与其他应用程序进行通信。

要使用 TIBEvents 组件，需要在 Database 属性中设置一个连接到 InterBase 服务器的 TIBDatabase 组件，然后可以使用 Events 属性来表示自己希望响应的事件。当数据库组件打开一个连接时，如果希望 TIBEvents 组件自动注册感兴趣的事件，则可以将其 AutoRegister 属性设置为 True；或者在打开一个数据库连接后，将 Registered 属性设置为 True。最后，可以在 OnEventAlert 事件处理器中编写一些代码来响应 InterBase 的一些事件行为。

##### 1. TIBEvents 组件主要的属性

- **AutoRegister** 如果该属性设置为 True，则数据库会自动注册该组件的事件，而不管其是否打开了一个到 InterBase 服务器的数据库连接。
- **Database** 设置连接到 InterBase 服务器的 TIBDatabase 组件。
- **Events** 该属性中可以列出 IBEvents 组件响应的事件。
- **Name** 设置 IBEvents 组件的名字。
- **Registered** 如果将该属性设置为 True，则会调用 RegisterEvents 方法来注册 Events 属性中列出的事件。

##### 2. TIBEvents 组件主要的方法

- **GetAutoRegister** 返回 AutoRegister 属性的值。

- RegisterEvents 注册 Events 属性中列出的感兴趣的事件。
- SetAutoRegister 设置 AutoRegister 属性的值, 其语法如下:  

```
procedure SetAutoRegister(const Value: Boolean);;
```

- UnRegisterEvents 不再注册 Events 列表中感兴趣的事件。

### 3. TIBEvents 组件主要的事件

- OnError 当 TIBEvents 组件在监视事件的过程中如果遇到一个异常, 则会触发该事件。
- OnEventAlert 当接收到一个 InterBase 事件时会触发该事件。

#### 5.1.12 TIBExtract 组件

TIBExtract 组件用于从 InterBase 数据库中提取元数据信息, 比如基表、视图、角色、索引及其他内容的一个列表。

可以通过如下方法使用 TIBExtract 组件:

- 设置 Database 和 Transaction 属性来指定提取信息及提供事务的数据库。
- 调用 ExtractObject 方法来指明希望提取的元数据的类型。
- 从 Items 属性中读取额外的信息。

### 1. TIBExtract 组件主要的属性

- Database 指定用于提取元数据的数据库, 即一个 TIBDatabase 组件。
- ShowSystem 该功能还没有实现, 为将来的版本保留。
- Transaction 指定从服务器提取元数据使用的事务。

### 2. TIBExtract 组件主要的方法

- ExtractObject 该方法用于从 InterBase 服务器中提取特定的元数据并将其写入到 Items 属性中。其语法如下:

```
procedure ExtractObject(ObjectType : TExtractObjectTypes; ObjectName : String = ''); ExtractTypes : TExtractTypes = []);
```

- GetArrayField 该方法返回数组字段的维数, 该字段的名称由 FieldName 参数指定。其语法如下:

```
function GetArrayField(FieldName : String) : String;
```

- GetCharacterSets 该方法查找特定的字符集和顺序, 并返回它们的名字。其语法如下:

```
function GetCharacterSets(CharSetId, Collation : Short; CollateOnly : Boolean) : String;
```

- GetFieldType 返回包括特定字段类型、子类型、伸缩性、大小、精度及长度的字符串描述。其语法如下:

```
function GetFieldType(FieldType, FieldSubType, FieldScale, FieldSize, FieldPrec, FieldLen : Integer) : String;
```

- Notification 当创建或销毁对象时会自动调用 Notification 方法, 以便 TIBExtract 可以进行必要的调整。

### 3. TIBExtract 组件主要的事件

该组件没有事件。

### 5.1.13 TIBClientDataSet 组件

该组件是一个客户数据集，它使用内部的 TIBDataSet 和 TDataSetProvider 组件来提取数据并应用更新。使用它可以缓存使用 InterBase Express 提取的数据，而不必使用一个外部的提供者和客户数据集。

使用 TIBClientDataSet 组件的缓存比其他组件可以提供更多的好处，包括以下方面：

- 既可以将 TIBClientDataSet 组件应用于从磁盘上文件提取数据的应用程序中，也可以用于从数据库服务器提取数据的应用程序中。从而可以实现公文包模式的应用程序。
- 可以利用唯一客户数据集的一些优点，比如维持集合、支持筛选及设定范围等。
- 可以很容易将使用该组件的应用程序移植为使用不同的访问机制的应用程序，因为 TIBClientDataSet 组件与 TSQLClientDataSet 及 TBDEClientDataSet 组件都非常相似。

TIBClientDataSet 在通过一个提供者连接到一个本地的 TIBDataSet 组件时，与 TClientDataSet 组件非常相似，只是其源数据集和提供者是内部的。它也提供了 TIBDataSet 和 TDataSetProvider 组件的一些属性，以便可以指定提取数据的数据库服务器，指明从服务器中提取什么样的数据，影响数据包中的何种信息，并在更新过程中提供输入。

除了使用源 TIBDataSet 组件外，TIBClientDataSet 组件还可以直接从一个磁盘文件上读写内容，内部提供者和源数据集不需要与基于文件的数据一起使用，如果要编写一个只使用文件的应用程序，则 TClientDataSet 是一种不错的选择。但是，从操作磁盘上文件的能力来看，TIBClientDataSet 组件最适合于公文包模式的应用程序。

#### 1. TIBClientDataSet 组件主要的属性

- Active 决定是否激活该数据集。
- Aggregates 列出应用到客户数据集的所有集合。
- AggregatesActive 指定该客户数据集是否可以计算及维持集合的值。
- CommandText 指定从数据库服务器中提取的数据。
- DBConnection 指定将该数据集连接到一个数据库服务器的组件。
- DBTransaction 指定管理该数据集使用的事务的事务对象。
- DisableStringTrim 指定在提交记录时是否删除字段值前后的空格。
- FileName 指定磁盘上的一个文件来保存客户数据集的数据。
- Filter 指定筛选的条件。
- Filtered 指定是否激活 Filter 属性中设置的筛选条件。
- IndexFieldNames 列出一些字段作为数据集的索引。
- IndexName 指定该客户数据集使用的索引。
- MasterFields 指定主表与从属表关联的字段，该属性只适用于从属表。
- MasterSource 该属性用于从属表中，用来指定与该表关联的主表使用的数据源。
- PacketRecords 该属性设置单几个数据包中记录的类型或数量，如果为-1，则表示包含所有的记录；如果是一个大于 0 的值，则表示该数据包包含记录的实际数量。
- Params 设置发送到提供者的参数值。

- **ReadOnly** 设置该客户数据集是只读的。
  - **UpdateMode** 指定应用记录更新的方式及范围。
2. **TIBClientDataSet** 组件主要的方法
- **CloneCursor** 共享属于另一个客户数据集的数据。其语法如下：
 

```
procedure CloneCursor(Source :TCustomClientDataSet; Reset: Boolean;
      KeepSettings: Boolean = False); virtual;
```
  - **GetQuoteChar** 返回在产生的 SQL 语句中使用的、用来包括引用字符串的一个或多个字符。
  - **Execute** 通过提供者数据集执行一个 SQL 命令。
  - **IsEmpty** 用来检验当前数据集是否包含记录。
  - **Open** 打开当前的数据集。
  - **Edit** 使当前的数据集进入编辑模式。
  - **Free** 销毁当前的数据集并释放其使用的内存。
3. **TIBClientDataSet** 组件主要的事件
- **After** 开头的事件 表示在某种情形出现之后触发该事件。
  - **Before** 开头的事件 表示在某种情形出现之前触发该事件。
  - **On** 开头的事件 表示在某种情形正在出现时触发该事件。

## 5.2 InterBase 面板组件应用实例

由于 InterBase 面板的组件也可用于建立跨平台的数据库应用程序，所以下面的实例都是使用 CLX 组件库建立的。这与使用 VCL 组件库的建立方法相同。

其实，使用 InterBase 面板的组件建立数据库应用程序与使用其他组件建立数据库应用程序的方法基本相同，但是有两点主要的不同：一是该面板的数据集本身无法直接连接到数据库，只能通过 **TIBDatabase** 组件连接到数据库；二是所有的数据集都需要使用一个 **IBTransaction** 组件来管理数据库的事务，而其他面板的数据库组件则不需要类似的组件。

### 5.2.1 使用 TIBTransaction 和 TIBTable 组件

该实例的实现步骤如下：

#### 1. 新建一个应用程序

通过 **File|New|CLX Application** 菜单新建立一个应用程序。

#### 2. 添加组件

(1) 选择 **InterBase** 面板，然后在表单 **Form1** 中添加一个数据库连接组件 **IBDatabase1**、一个数据库事务组件 **IBTransaction1**、一个基表组件 **IBTable1**，再通过 **Data Access** 面板添加一个数据源组件 **DataSource1**。

(2) 选择 **Data Controls** 面板，然后在 **Form1** 中添加一个数据表格组件 **DBGrid1** 和一个数据导航组件 **DBNavigator1**。

#### 3. 设置组件的属性

各个组件的属性设置值见表 5-4。

表 5-4 设置各个组件的属性

组件	属性	设置值	说明
IBDatabase1	DatabaseName	D:\Borland6\Borland Shared\Data\Employee.gdb	设置数据库的名字，包括完整的路径。Delphi 6 的安装路径不同，该数据库的实际位置可能不同
	DefaultTransaction	IBTransaction1	设置数据库默认的事务，这样使用该数据库连接组件的其他组件会自动设置该事务
IBTransaction1			使用各个属性的默认值即可
IBTable1	Active	True	激活数据集
	Database	IBDatabase1	设置数据库连接组件
	Filter	emp_no<14	设置记录筛选条件
	Filtered	True	激活筛选器
	TableName	EMPLOYEE	设置基表的名字
	Transaction	IBTransaction1	设置使用的默认事务
DataSource1	DataSet	IBTable1	设置数据源使用的数据集
DBGrid1	DataSource	DataSource1	设置使用的数据源
DBNavigator1	DataSource	DataSource1	设置使用的数据源

以上组件的属性设置完后，Form1 的设计界面如图 5-1 所示。如果运行程序，其界面与设计界面相似。

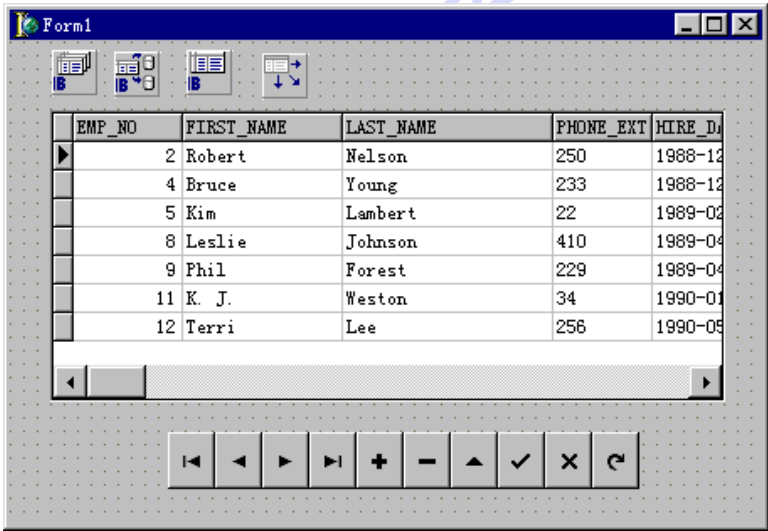


图 5-1 Form1 的设计界面

5.2.2 使用 TIBDatabaseInfo 组件获得数据库信息

InterBase 组件面板提供的 TIBDatabaseInfo 组件可以显示当前连接的一些数据库信息。



这个实例就通过该组件显示当前连接的数据库的一些信息。

该实例的实现步骤如下：

### 1. 新建一个应用程序

通过 File|New|CLX Application 菜单新建立一个应用程序。

### 2. 添加组件

(1) 选择 InterBase 面板，然后在表单 Form1 中添加一个数据库连接组件 IBDatabase1、一个数据库事务组件 IBTransaction1、一个数据库信息组件 IBDatabaseInfo1、一个基表组件 IBTable1，再通过 Daba Access 面板添加一个数据源组件 DataSource1。

(2) 选择 Standard 组件面板，在 Form1 中添加一个多行文本编辑框 Memo1。再选择 Additional 组件面板，选择 TTimer 组件，在表单中添加一个定时器 Timer1。该定时器的作用是定期刷新 Memo1 中显示的数据库信息。

(2) 选择 Data Controls 面板，然后在 Form1 中添加一个数据表格组件 DBGrid1 和一个数据导航组件 DBNavigator1。

### 3. 设置组件的属性

各个组件属性的设置值见表 5-5。

表 5-5 设置各个组件的属性

组件	属性	设置值	说明
IBDatabase1	DatabaseName	D:\Borland6\Borland Shared\Data\Employee.gdb	设置数据库的名字，包括完整的路径。Delphi 6 的安装路径不同，该数据库的实际位置可能不同
	DefaultTransaction	IBTransaction1	设置数据库默认的事务，这样使用该数据库连接组件的其他组件会自动设置该事务
IBTransaction1			使用各个属性的默认值即可
IBDatabaseInfo1	Database	IBDatabase1	设置显示信息的数据库
Timer1	Interval	2000	设置刷新间隔为 2000 毫秒
IBTable1	Active	True	激活数据集
	Database	IBDatabase1	设置数据库连接组件
	TableName	CUSTOMER	设置基表的名字
	Transaction	IBTransaction1	设置使用的默认事务
DataSource1	DataSet	IBTable1	设置数据源使用的数据集
DBGrid1	DataSource	DataSource1	设置使用的数据源
DBNavigator1	DataSource	DataSource1	设置使用的数据源

以上组件的属性设置完后，Form1 的设计界面如图 5-2 所示。

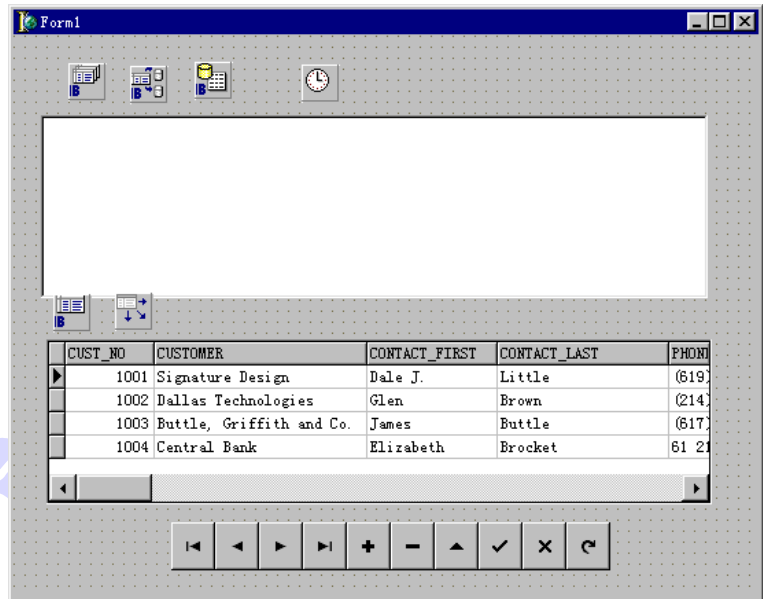


图 5-2 Form1 的设计界面

#### 4. 添加代码

在 Timer1 上面双击鼠标，打开定时器的 OnTimer 事件处理器，添加显示数据库信息的代码，并将这些信息显示在 Memo1 中。完整的代码如下：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Memo1.Text := '数据库文件名:'+IBDatabaseInfo1.DBFileName+';'+#13#10;
    Memo1.Text := Memo1.Text+'数据库站点名:'+IBDatabaseInfo1.DBSiteName
    +';'+#13#10;
    Memo1.Text := Memo1.Text+'路径
名:'+IBDatabaseInfo1.GetNamePath+';'+#13#10;
    Memo1.Text := Memo1.Text+'用户
名:'+IBDatabaseInfo1.UserNames.Strings[0]+';'+
    +#13#10;
    Memo1.Text := Memo1.Text+'删除记录数量
'+copy(IBDatabaseInfo1.DeleteCount.Text,
    ,4,4) +';'+#13#10;
    Memo1.Text := Memo1.Text+'插入记录数量'+copy
    (IBDatabaseInfo1.InsertCount.Text
    ,4,4) +';'+#13#10;
    Memo1.Text := Memo1.Text+'修改记录数量
'+copy(IBDatabaseInfo1.UpdateCount.Text
    ,4,4) +';'+#13#10;
end;
```

以上的#13#10 表示回车换行符，这是 Delphi 表示特殊字符的一种方法。即用符号#加上一个字符的 ASCII 值来表示该字符。另外，copy 方法是一个字符串函数，用来拷贝字符串中部分内容，因为 UpdateCount 等返回的记录数量总是有一个 134 的数字，通过该方法可以删除这几个字符。另外，以上通过字符串连接运算符加号“+”，将所有信息都连接成

一个字符串放置到 Memo1 中。

### 5. 运行程序

下面运行程序。然后分别在数据表格中插入一条记录、修改两条记录，此时运行的结果如图 5-3 所示。

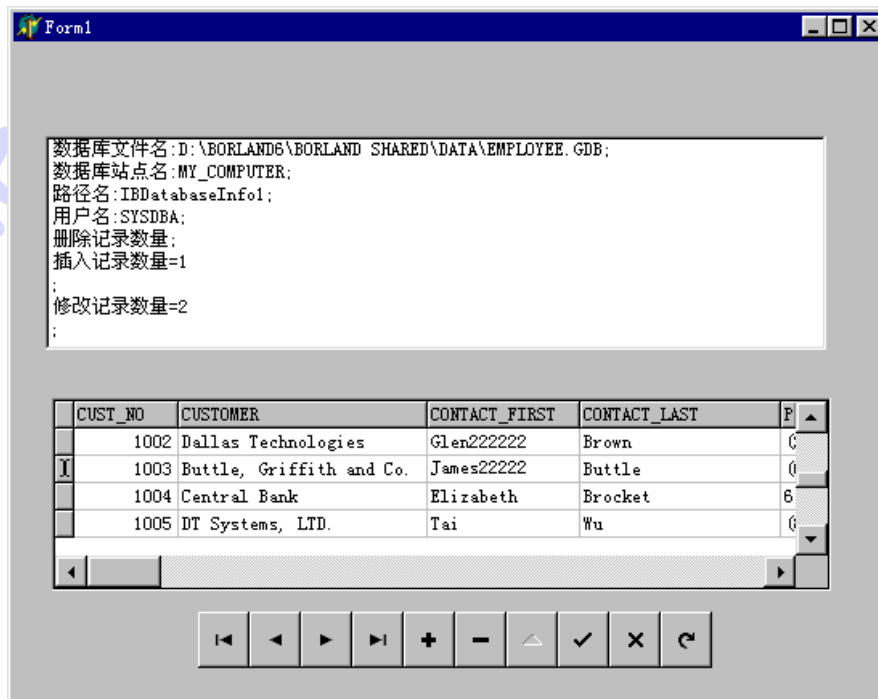


图 5-3 修改及添加记录后界面的显示

## 第6章 dbExpress 开发跨平台数据库程序

dbExpress 是 Delphi 6 中新添加的功能，其主要功能都是通过 dbExpress 组件面板实现的，这些组件可以实现跨平台数据库应用程序开发，因为在 Linux 下的 Kylix 提供了相同的组件及功能。在 ADO 及 BDE 面板上都能找到类似的对应组件。使用 dbExpress 组件面板上的组件，可以快速访问数据库的信息。当然，InterBase 组件面板也可开发跨平台程序。

### 6.1 dbExpress 简介

dbExpress 是一个跨平台的、不依赖于数据库的一个独立的层，它提供了一些方法用于动态 SQL 语句的处理。它定义了一个通用的接口用于访问不同的 SQL 服务器，并为各种数据库提供了驱动程序，这些驱动程序以独立动态库的形式存在，用于实现通用的 dbExpress 接口来处理查询和存储过程。这些驱动程序在 Windows 和 Linux 下都可以使用，在 Windows 下是动态库“.dll”文件，在 Linux 下是共享对象“.so”文件。

由于将 dbExpress 设计为一种快速、简便并且容易发布的对象，所以它只具备有限的数据处理功能，主要作用是作为数据库服务器客户端软件的外壳程序出现的。它提供了通用 API 的许多优点，而没有 BDE 等数据库引擎需要的额外开销。例如，dbExpress 驱动程序只返回单方向的游标，并且不能执行数据或元数据的缓存。由于保持核心运行时数据库访问层的简单和轻便，所以 dbExpress 提供了高性能的数据库连接，可以非常容易地适应新的数据源。

通过 dbExpress 可以开发标准的执行文件，如果文件是独立运行的，则这些可执行文件中必须静态链接上 dbExpress 对象文件，即一些.dcu 文件，各种.dcu 文件与数据库的对应关系，见表 6-1。

表 6-1 各种.dcu 文件与数据库的对应关系

文件名	作用
DBExpInt.dcu	连接到 InterBase 数据库的应用程序
DBExpOra.dcu	连接到 Oracle 数据库的应用程序
DBExpDb2.dcu	连接到 DB2 数据库的应用程序
DBExpMy.dcu	连接到 MySQL 数据库的应用程序
Crtl.dcu、MidasLib.dcu	使用客户数据集的 dbExpress 可执行文件需要这些文件，比如 TSQLClientDataSet 客户数据集

如果开发的应用程序不是独立运行的文件，则可以将该执行文件与 dbExpress 驱动程序和 DataSnap DLL 关联，各种动态库的作用，见表 6-2。

表 6-2 各种动态库的作用

文件名	作用
DBExpInt.dll	连接到 InterBase 数据库应用程序
DBExpOra.dll	连接到 Oracle 数据库应用程序
DBExpDb2.dll	连接到 DB2 数据库应用程序
DBExpMy.dll	连接到 MySQL 数据库应用程序
Midas.dll	使用客户数据集的数据库应用程序需要

## 6.2 dbExpress 面板组件介绍

该面板包括的组件可以建立使用 dbExpress 的数据库应用程序。各个组件的作用如下：

- **SQLConnection** 建立到数据库服务器的连接。
- **SQLDataSet** 使用 dbExpress 的一般数据集。
- **SQLQuery** 使用 dbExpress 及 SQL 语句建立的数据集。
- **SQLStoredProc** 使用 dbExpress 及存储过程建立的数据集。
- **SQLTable** 使用 dbExpress 访问数据库基本表的数据集。
- **SQLMonitor** 截取一个 SQL 连接组件和数据库服务器之间传递的信息，并将其保存到一个字符串列表中。
- **SQLClientDataSet** 该组件作为一个客户数据集，可以缓存内存中的信息并保存应用程序进行的任何修改。使用一个内部的 TSQLDataSet 和 TDataSetProvider 组件可以提取数据并应用更新。

由于该面板的组件既有 VCL 版本（用于 Windows），也有一个 CLX 版本（可用于跨平台），而两者的大部分属性、方法和事件都相似。所以下面只讲述 CLX 组件，以便将这些组件用于建立跨平台的数据库应用程序。

### 6.2.1 TSQLConnection 组件

与 BDE 和 ADO 等面板的数据集组件不同，dbExpress 面板的数据集组件只能通过 TSQLConnection 组件连接到数据库，而不能直接连接到数据库。通过设置数据集组件的 SQL Connection 属性，它可以被多个 SQL 数据集组件共享。

TSQLConnection 组件与 dbExpress 驱动程序和 dbxdrivers.ini 及 dbxconnections.ini 两个文件交互，这两个文件保存在...\Borland Shared\DBExpress\目录下。dbxdrivers.ini 文件列出了安装的数据库驱动程序类型，并且对于每种驱动程序列出了其使用的动态库（Windows 下）及共享对象（Linux 下），它也需要所有连接参数的默认设置。dbxconnections.ini 文件中列出了已命名的多套连接配置，每种配置代表了一系列的 TSQLConnection 连接配置，并且描述了一种特定的数据库连接。

在使用 TSQLConnection 建立数据库连接时，可以从 dbxconnections.ini 文件中选择一种已经定义的配置，也可以在该文件中定义一些新的配置。可以在数据模块或表单中双击 TSQLConnection 组件打开数据库连接配置面板进行新的设置。

#### 1. TSQLConnection 组件主要的属性





图 6-1 的 Connection Name（连接名字）列表中单击鼠标右键打开的上下文菜单作用相同。

◇  新建一个数据库连接。它会打开一个对话框,如图 6-3 所示。其中 Driver Name 一栏用于选择一种数据库驱动程序;在 Connection Name 一栏,可以为新建立的数据库连接输入一个有意义的名字。

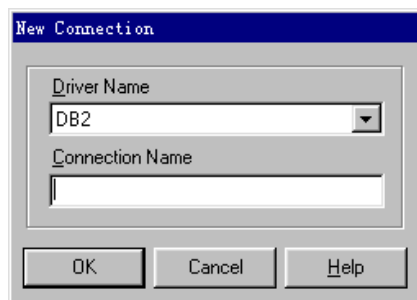






图 6-3 新建一个数据库连接

◇  删除一个数据库连接。方法是在图 6-1 左边的 Connection Name 列表框中选择一个要删除的数据库连接,然后单击该按钮即可删除它。

◇  重命名一个数据库连接。与删除数据库连接的方法相似。

◇  测试一个数据库连接。先在 Connection Name 列表中选择一连接,然后单击该按钮,如果已经正确设置了该数据库连接,一般会弹出一个用户注册窗口,输入正确的用户名及口令,就可以测试是否可以成功连接到数据库。

◇  单击该按钮,会打开一个面板,如图 6-4 所示,该面板列出了 Delphi 提供的 dbExpress 驱动程序及对应的库名字和供应商的动态库等内容。

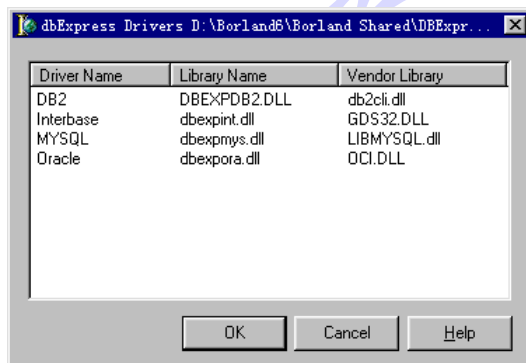


图 6-4 dbExpress 驱动程序及对应的库

- **Connected** 如果该属性为 True,则表示激活当前的数据库连接。
- **DriverName** 设置数据库驱动程序的名字。该属性一般不要手工设置,因为设置 ConnectionName 属性时,会根据当前使用的数据库自动选择其属性值。
- **DataSets** 列出当前数据库连接组件所有活动的数据集,其范围是从 0 到返回一个数值 DataSetCount,其语法如下:

```
property DataSets[Index: Integer]: TCustomSQLDataSet;
```

- **DriverName** 指定与 SQL 连接关联的数据库驱动程序。常用的 dbExpress 驱动程

序有 INTERBASE、MYSQL、ORACLE 或 DB2。

- **GetDriverFunc** 指定返回顶级 dbExpress 驱动程序的函数。
- **KeepConnection** 在没有数据集打开的情况下，指定该连接是否保持活动状态。
- **LibraryName** 指定 TSQLConnection 组件用于建立连接的 dbExpress 驱动程序。
- **LoadParamsOnConnect** 如果希望在连接到一个服务器前，TSQLConnection 组件装载 ConnectionName 中设定的配置，则可以将该属性设置为 True。
- **MaxStmtsPerConn** 获得一个数据库连接可以支持的语句的最大数量。
- **MetaData** 为提供服务器元数据的 dbExpress 对象设置一个接口。但是，一般情况下，不能直接通过该属性来获得元数据，而一般使用下面的方法：比如 GetFieldNames、GetIndexNames、GetProcedureNames、GetProcedureParams 或 GetTableNames。如果需要更详细的元数据，则可以 SQL 数据集的 SetSchemaInfo 方法。
- **Params** 列出了连接参数。参数的形式一般为：Name=Value，前面是参数的名字，后面是参数的值。可以使用类似下面的语句添加参数值：  

```
Params.Values['User_Name'] := 'scott';
```
- **ParamsLoaded** 不要使用该参数。因为它是被“连接编辑器”内部使用。
- **SQLConnection** 为代表一个连接的 dbExpress 对象提供一个接口。
- **SQLHourGlass** 指定光标在过长的操作中是否需要变为等待的形状。
- **TableScope** 在取基表的计划信息时，指示何种类型的基表可以返回。可选值及其作用，见表 6-3。

表 6-3 各种设置值及其作用

属性选择值	作用
tsSynonym	同义次
tsSysTable	系统表
tsTable	一般基表
tsView	视图

- **VendorLib** 指定数据库供应商提供的客户软件库（DLL 库或共享对象）。一般应用程序不需要设置该属性，但是当设置 DriverName 属性时，会自动设置该属性。

2. TSQLConnection 组件主要的方法

- **CloneConnection** 调用该方法可以建立当前 SQL 连接的一个副本。
- **CloseDataSets** 关闭与当前连接关联的所有数据集。
- **Commit** 提交一个打开的事务。
- **Execute** 在服务器上运行一个 SQL 命令。其语法如下：

```
function Execute(const SQL: string; Params: TParams;  
    ResultSet: Pointer=nil): Integer;
```

其中，SQL 是执行的命令语句；Params 是 SQL 语句使用的参数；ResultSet 是指向 TCustomSQLDataSet 类型变量的指针。如果 SQL 语句没有参数，则可以使用 ExecuteDirect 方法代替该方法。

- **ExecuteDirect** 执行没有参数的 SQL 命令，其语法如下：  

```
function ExecuteDirect(const SQL: string): LongWord;
```
- **GetFieldNames** 获得一个基表中所有字段的列表。其语法如下：  

```
procedure GetFieldNames(const TableName: String; List: TStrings);
```
- **GetIndexNames** 获得一个基表中所有索引的列表。其语法与 **GetFieldNames** 相似。
- **GetProcedureNames** 获得关联数据库中所有存储过程的列表。语法如下：  

```
procedure GetProcedureNames(List: TStrings);
```
- **GetProcedureParams** 获得特定存储过程中有关参数的一些信息。
- **GetTableNames** 获得关联数据库中基表的名字列表，语法如下：  

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```
- **LoadParamsFromIniFile** 用于设置与 **ConnectionName** 属性关联的 **DriverName** 和 **Params** 属性的值。
- **Rollback** 用于结束一个事务，并取消该事务中进行的任何改变。
- **StartTransaction** 在一个数据库中开始一个新的事务。

### 3. TSQLConnection 组件主要的事件

该组件共有 5 个事件，只有 **OnLogin** 事件是特有的，其他事件继承自 **TcustomConnection** 组件。下面只说明 **OnLogin** 组件。

- **OnLogin** 当 **TSQLConnection** 组件试图连接到数据库服务器时，触发该事件。使用该事件可以向 **User\_Name**、**Password** 和 **Database** 参数中分配一些值。

#### 6.2.2 TSQLDataSet 组件

该组件是一个通用的单方向的数据集，用于使用 **dbExpress** 访问数据库信息。在设计阶段可以将一个 **TSQLDataSet** 组件添加到一个表单或数据模块中，或者在运行时动态建立一个。

**TSQLDataSet** 组件可以表示一个数据库基表的多条记录、一个 **SELECT** 查询的结果集或一个存储过程返回的结果集；也可以运行没有结果集返回的一个查询或存储过程；还可以表示数据库服务器上可用的元数据，比如基表、存储过程、一个基表中的字段等。

由于 **TSQLDataSet** 组件是一种单方向的数据集，所以不能像其他数据集那样在内存中缓存多条记录，也因此只能使用 **First** 及 **Next** 方法导航，同时不提供内置的编辑功能，即无法对数据直接进行删除、修改、插入等操作。要进行编辑，只能在一个 SQL 数据集中显式地建立一个 SQL **UPDATE** 命令，或将当前数据集连接到使用提供者的一個客户数据集。**TSQLDataSet** 组件也不提供诸如筛选器或搜索字段等需要缓存多条记录的命令。

与其他类型的数据集不同，**TSQLDataSet** 组件必须通过设置 **SQLConnection** 属性来建立数据库的连接，自己不能直接连接到数据库。

#### 1. TSQLDataSet 组件主要的属性

- **Active** 决定是否激活一个数据集。
- **CommandType** 选择 SQL 命令的类型，可以是基表 (**ctTable**)、查询 (**ctQuery**) 及存储过程 (**ctStoredProc**)。
- **CommandText** 设置命令文本，会根据 **CommandType** 中设置值不同而选择不同的

类型。

- **DataSource** 设置另一个数据集，以便可以使用该数据集中的字段作为该数据集参数值的提供者。一般不需要设置该属性。
- **MaxBlobSize** 该属性用于限制可以从 BLOB (Binary Large Object, 二进制大对象) 字段提取的数据的数量限制。设置为 0 表示可以取任意大的。
- **NoMetadata** 指定 SQL 数据集是否根据数据提取元信息。
- **ParamCheck** 如果该属性为 True, 则当 SQL 命令改变时, 一个 SQL 数据集的参数列表会重新产生。
- **Params** 用于设置一个查询或存储过程的参数。
- **RecordCount** 获得与数据集关联的记录总数。
- **SortFieldNames** 如果 CommandType 属性的值是 ctTable, 该属性可以指定一些字段作为数据排序的依据。
- **SQLConnection** 指定将数据集连接到一个数据库服务器的 SQL 连接组件。
- **TransactionLevel** 指明当前数据集隶属于哪一个事务。

## 2. TSQLDataSet 组件主要的方法

- **ExecSQL** 执行不返回一系列记录的一个查询或存储过程。
- **Edit** 使数据集的数据进入可编辑状态。
- **Open** 打开当前的记录集。

## 3. TSQLDataSet 组件主要的事件

- **AfterClose** 当应用程序关闭该数据集时触发该事件。
- **AfterOpen** 当应用程序打开该数据集而没有开始访问数据时触发该事件。
- **OnCalcFields** 当应用程序重新计算可计算字段时触发该事件。

### 6.2.3 TSQLQuery 组件

TSQLQuery 组件用于在数据库服务器上运行一个 SQL 命令, 它要通过 TSQLConnection 连接到数据库。TSQLQuery 组件可以表示一个使用 SELECT 语句建立的结果集, 也可以执行一些没有结果集的命令语句, 例如 INSERT、DELETE、UPDATE、ALTER TABLE 等。即可以在设计阶段使用 TSQLQuery 组件, 也可以在运行时动态建立一个 TSQLQuery 组件。

TSQLQuery 组件是一种单方向的数据集, 不能在内存中缓存数据, 所以只能使用 First 及 Next 方法进行导航, 而不能使用其他数据集常用的导航方法。它不支持对数据的直接编辑, 而只能通过其他方式实现, 比如显式地建立一个 SQL UPDATE 命令, 或使用一个提供者将数据集连接到客户, 但不支持记录筛选及搜索字段等需要缓存多条记录的功能。

## 1. TSQLQuery 组件主要的属性

- **Active** 确定是否激活当前数据集。
- **DataSource** 将当前数据集链接到另一个数据集。一定不要使用当前数据集的数据源设置该属性。
- **MaxBlobSize** 设置 BLOB 字段可以提取的数据的最大字节数, -1 表示没有限制。
- **Name** 设置当前数据集的名字。
- **ObjectView** 确定字段是否以多层次或单层形式存储在 Fields 属性中。

- ParamCheck 确定在 SQL 命令改变时, 是否重新建立 SQL 数据集的参数列表。
- Params 设置一个查询或存储过程的参数。
- SQL 指定运行在数据库服务器上的 SQL 语句。
- SQLConnection 用于设置一个 TSQLConnection 组件来连接到数据库服务器。
- Text 表示 SQL 语句以单个字符串的形式出现。

## 2. TSQLQuery 组件主要的方法

- ExecSQL 该方法用于运行不返回一个结果集的查询, 此时不能使用 SELECT 语句, 而只能使用 INSERT、UPDATE、DELETE 或 CREATE TABLE 语句。其语法如下:

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

- ParamByName 确定 Params 属性中的一个参数, 其语法如下:

```
function ParamByName(const Value: string): TParam;
```

其中, Value 是该参数的名字。

- Close 关闭当前的数据集。
- Edit 设置当前数据集处于编辑状态。
- Delete 删除当前活动记录, 并将指针移动到下一条记录。
- Insert 在记录集当前位置插入一条新的空记录。
- IsEmpty 检验当前数据集是否为空。
- First 移动到数据集的第一条记录。
- Next 移动到数据集的下一条记录。
- Open 打开当前数据集。
- Free 销毁当前数据集, 并释放其使用的内存。

## 3. TSQLQuery 组件主要的事件

- AfterClose 在关闭数据集后触发该事件。
- AfterOpen 在打开数据集后触发该事件。
- AfterRefresh 在刷新数据集后触发该事件。
- AfterScroll 在从一条记录滚动到另一条记录后触发该事件。
- BeforeClose 在关闭数据集前触发该事件。
- BeforeOpen 在打开数据集前触发该事件。
- BeforeRefresh 在刷新数据集前触发该事件。
- BeforeScroll 在从一条记录滚动到另一条记录前触发该事件。
- OnCalcFields 当打开一个数据集、数据集进入可编辑 (dsEdit) 状态、从数据库中提取一个记录时触发该事件。如果 AutoCalcFields 属性为 True 时, 如果输入焦点在不同控件、同一控件的不同记录或不同字段及记录进行修改时, 都会触发该事件。

### 6.2.4 TSQLStoredProc 组件

TSQLStoredProc 组件用来运行数据库服务器上的一个存储过程, 它要通过 TSQLConnection 组件连接到数据库。TSQLStoredProc 组件可以返回一个结果集, 也可以不



返回一个结果集。

该数据集与 TSQLQuery 数据集一样，也是一个单方向的数据集，所以具有单方向数据集的一些特点，比如导航及编辑都受到一定的限制，不能缓存多条记录等。

#### 1. TSQLStoredProc 组件主要的属性

- Active 确定是否激活当前数据集。
- MaxBlobSize 设置 BLOB 字段可以提取的数据的最大字节数，-1 表示没有限制。
- Name 设置当前数据集的名字。
- ObjectView 确定字段是否以多层或单层形式存储在 Fields 属性中。
- ParamCheck 确定在 SQL 命令改变时，是否重新建立 SQL 数据集的参数列表。
- Params 设置一个查询或存储过程的参数。
- SQLConnection 用于设置一个 TSQLConnection 组件来连接到数据库服务器。
- StoredProcName 指定作为该数据集来源的存储过程的名字，该存储过程保存在数据库服务器中。

#### 2. TSQLStoredProc 组件主要的方法

- ExecProc 运行一个不返回一个游标的存储过程。
- NextRecordSet 该方法用于访问第二个记录集。
- Close 关闭当前的数据集。
- Edit 设置当前数据集处于编辑状态。
- Delete 删除当前活动记录，并将指针移动到下一条记录。
- Insert 在记录集当前位置插入一条新的空记录。
- IsEmpty 检验当前数据集是否为空。
- First 移动到数据集的第一条记录。
- Next 移动到数据集的下一条记录。
- Open 打开当前数据集。
- Free 销毁当前数据集，并释放其使用的内存。

#### 3. TSQLStoredProc 组件主要的事件

- AfterClose 在关闭数据集后触发该事件。
- AfterOpen 在打开数据集后触发该事件。
- BeforeClose 在关闭数据集前触发该事件。
- BeforeOpen 在打开数据集前触发该事件。

### 6.2.5 TSQLTable 组件

TSQLTable 与 TTable 的作用相似，都是访问数据库的基表，不过 TSQLTable 使用的是 dbExpress，而 TTable 却使用 BDE。两者还有以下几个重要区别：

(1) TSQLTable 是一个单方向的数据集，该数据集无法使用内存缓存多条记录，所以只能使用 First 和 Next 方法导航，不能直接编辑数据，只能显式地使用一个 SQL UPDATE 命令，或者使用一个提供者将基表连接到一个客户数据集，来实现对数据的编辑；而 TTable 却具备导航、编辑及缓存多记录的功能。



(2) TSQLTable 需要通过数据库连接组件 TSQLConnection 才能连接到数据库；而 TTable 可以通过数据库连接组件 TDatabase 连接到数据库，也可以直接连接到数据库。

#### 1. TSQLTable 组件主要的属性

- **Active** 该属性设置为 True 时，会激活该数据集。
- **IndexFieldNames** 该属性中可以输入一个或多个字段的名称用于记录的排序，多个字段之间使用分号 “;” 间隔。另外，这些字段还可以作为主从表应用程序中从属表连接到主表的关联字段。
- **IndexName** 选择一个已经在数据库中定义的索引来进行记录的排序，并作为从属表连接到主表的关联字段。
- **MasterFields** 该属性只用于主从表应用程序从属表的数据集中，设置连接到主表的字段。
- **MasterSource** 设置主表使用的数据源。
- **MaxBlobSize** 设置 BLOB 字段可以使用的最大字节数。如果是 -1，则表示没有最大值的限制。
- **Name** 该数据集的名称。
- **ObjectView** 指定数据集的字段在 Fields 属性中是以多层次存储，还是以相同的层次存储。该属性的设置会影响 TDBGrid 组件显示 AD 及数组字段的方式。
- **SQLConnection** 指定将该数据集连接到数据库服务器的 SQL 连接组件。
- **TableName** 指定使用的基表的名称。
- **Bof** 表示当前记录指针指向第一条记录。
- **CanModify** 由于当前数据集是单方向的，所以该属性的设置值总是 False。
- **DataSource** 用于将当前数据集连接到另一个数据集。
- **Eof** 表示当前记录指针指向最后一条记录。
- **IsUnidirectional** 可通过该属性确定当前数据集是否为单方向的。
- **RecordCount** 获得当前数据集中记录的数量。
- **State** 决定该数据集的操作模式。

#### 2. TSQLTable 组件主要的方法

- **DeleteRecords** 删除数据库基表中所有的记录。
- **GetIndexNames** 该方法可以获得在数据库中为当前及已定义的所有索引的名称列表，其语法如下：

```
procedure GetIndexNames(List: TStrings);
```

- **PrepareStatement** 该方法会产生一个 SELECT 语句，从数据库的当前基表中提取数据。注意，在应用程序中不要直接调用该方法，需要时将 Prepared 属性设置为 True 即可。

- **ParamByName** 设置 Params 属性中设置的各个参数的值，其语法如下：

```
function ParamByName(const Value: string): TParam;
```

其中，Value 表示该参数的名称。该方法主要用于运行时。下面是使用该方法的例子：

```
SQLDataSet1.Params.CreateParam(ftString, 'PartNo', ptInput); //建立参数
SQLDataSet1.ParamByName('PartNo').Value := Edit1.Text; //设置参数值
```

```

SQLDataSet1.Prepared := False;           //为查询准备好新参数
SQLDataSet1.Close;                       //关闭数据集
SQLDataSet1.Open;                       //打开数据集

```

- Close 关闭数据集。
- Delete 删除当前活动记录，并将指针移动到下一条记录。
- Edit 将当前数据集设置为可编辑状态。
- First 移动到数据集的第一条记录。
- Next 移动到数据集的下一条记录。
- Open 打开当前数据集。
- Free 销毁当前数据集，并释放其使用的内存。

### 3. TSQLTable 组件主要的事件

- AfterClose 在关闭数据集后触发该事件。
- AfterOpen 在打开数据集后触发该事件。
- AfterRefresh 在刷新数据集后触发该事件。
- AfterScroll 在从一条记录滚动到另一条记录后触发该事件。
- Before 开头的事件 这些事件与上面的 4 个事件对应，只是发生在这些情形以前。
- OnCalcFields 当打开一个数据集、数据集进入可编辑（dsEdit）状态、从数据库中提取一个记录时触发该事件。AutoCalcFields 属性为 True 时，输入焦点在不同控件、同一控件的不同记录或不同字段及记录进行修改时，都会触发该事件。

#### 6.2.6 TSQLMonitor 组件

TSQLMonitor 组件用于调试应用程序和数据库服务器之间的通信，该组件会使用一个字符列表来记录来自一个特定 SQL 连接组件的 SQL 命令，这样不但可以显示出显式添加到 SQL 数据集或 SQL 连接组件中的命令，也会显示出应用程序自动产生的命令，比如一个数据提供者产生的命令。可以使用 TraceList 访问该字符串列表中的命令。当连接组件向数据库服务器传递信息时，会自动更新 TraceList 属性。

注意，TSQLMonitor 组件使用 SQL 连接组件的 TraceCallbackEvent 属性来监视该连接组件的消息。如果自己对该属性进行了调用，则 SQL 监视实例将无法起作用。

### 1. TSQLMonitor 组件主要的属性

- Active 该属性用于决定监视器是否开始录制传递到数据库服务器的 SQL 命令。默认值 False 表示停止记录。
- AutoSave 指定被监视的消息是否自动保存到一个文件中。默认值 False 表示不能自动保存。
- FileName 指定保存被监视消息的文件。
- Name 指定监视器实例的名字。
- SQLConnection 指定连接到数据库的连接组件，该组件的消息将被监视。
- TraceList 该属性用于列出在 SQL 连接组件和数据库服务器之间传递的消息。

### 2. TSQLMonitor 组件主要的方法

- LoadFromFile 从一个文件中提取字符串放置在 TraceList 属性中。其语法如下：

```
procedure LoadFromFile(AFileName: string);
```

- **SaveToFile** 将 **TraceList** 属性中的内容保存到一个文件中，其语法如下：

```
procedure SaveToFile(AFileName: string);
```

例如，下面例子将内容保存到 **mylog.txt** 文件中：

```
procedure TForm1.SQLMonitor1LogTrace(Sender: TObject; CBInfo: Pointer);
begin
    TSQLMonitor(Sender).SaveToFile('c:\mylog.txt');
end;
```

### 3. TSQLMonitor 组件主要的事件

- **OnLogTrace** 当一条新的消息记录到列表中时触发该事件。
- **OnTrace** 当一条消息被检测到但是还没有登录到列表以前触发该事件。

#### 6.2.7 TSQLClientDataSet 组件

**TSQLClientDataSet** 是一个客户数据集，它使用内部的 **TSQLDataSet** 和 **TDataSetProvider** 组件来提取数据，它结合了快速访问很容易开发一个单方向的数据集，并且使用一个客户数据集的能力来编辑及导航数据。

由于它是一个内部的 **TSQLDataSet**，所以 **TSQLClientDataSet** 可以使用 **dbExpress** 快速访问数据库的信息。并且因为它是一个客户数据集，所以它可以在内存中缓存信息而保存应用程序进行的任何数据更新。由于它有一个内部提供者，所以它可以将这些更新重新保存到数据库中。当 **TSQLClientDataSet** 通过一个提供者连接到本地的 **TSQLDataSet** 时，它与 **TClientDataSet** 的工作机制非常相似，只不过它的源数据集和提供者都是内部的。

**TSQLClientDataSet** 结合了 **TSQLDataSet** 和 **TDataSetProvider** 组件的一些属性及事件，可以指定提取数据的数据库服务器、指明从服务器中提取的数据内容、影响哪些数据包中的信息，并且在更新过程中提供输入。

除了使用源 **TSQLDataSet**，**TSQLClientDataSet** 可以读写磁盘上一个专用文件，内部提供者和源数据集不需要使用基于文件的数据。如果要编写纯粹基于文件的应用程序，则可以使 **TClientDataSet** 满足这种选择，但是，读写磁盘上文件的能力使得 **TSQLClientDataSet** 非常适合于公文包模型的应用程序。

**注意：**一般情况下不要将 **TSQLClientDataSet** 应用于主从表关联的应用程序中，因为它不能优化查询，并且会导致非常低的性能。

#### 1. TSQLClientDataSet 组件主要的属性

- **Active** 决定是否激活该数据集。
- **Aggregates** 列出所有应用于客户数据集的集合。
- **AggregatesActive** 决定该客户数据集是否计算及维持集合的值。
- **AutoCalcFields** 决定何时触发 **OnCalcFields** 事件及何时计算搜索字段的值。
- **CommandText** 指定如何从数据库服务器中提取数据。该属性会根据 **CommandType** 属性设置值的不同而使用不同的内容。如果 **CommandType** 是 **ctQuery**，则该属性要输入一条可执行的 SQL 语句；如果 **CommandType** 是 **ctTable**，则该属性是一个基表的名字；如果 **CommandType** 是 **ctStoredProc**，则该属性是一

个存储过程的名字

- **CommandType** 用于确定 CommandText 属性的意义。该属性可以设置的三个值分别是 ctQuery、ctTable 和 ctStoredProc。
- **ConnectionName** 指定连接到数据库服务器使用的数据库连接的名称。可以双击 TSQLConnection 组件重新编辑或添加新的数据库连接名称。
- **Constraints** 设置编辑数据时需要的记录级的限制。
- **DBConnection** 设置一个将该数据集连接到数据库服务器的 TSQLConnection 组件。
- **DisableStringTrim** 当该属性设置为 False 时, 记录中的字段内容要提交到数据库中时, 会删除两边的多余空格。
- **FetchOnDemand** 指明是否可以根据需要从提供者那里提取数据包。
- **FieldDefs** 指出为数据集定义的字段列表。
- **FileName** 指定一个保存该客户数据集数据的文件。
- **Filter** 用于设置筛选条件的具体内容。
- **Filtered** 用于决定 Filter 属性中设置的内容是否起作用。
- **FilterOptions** 指定按照大小写筛选数据, 并且筛选记录时是否可以局部比较。
- **IndexDefs** 包含一个客户数据集的索引信息。
- **IndexFieldNames** 列出多个字段作为数据显示的索引。
- **IndexName** 设置一个在数据库服务器中已经定义的一个索引。
- **MasterFields** 为了建立主从关联关系, 列出主表中的多个字段来链接该数据集中的相应字段。
- **MasterSource** 设置主表对应的数据源。该属性只能用于主从关联表中。
- **Options** 这些选项设置如何提取及使用数据。
- **PacketRecords** 指明在单个数据包中记录的数量和类型。
- **Params** 包含发送到提供者的参数值。
- **ReadOnly** 指明客户数据集对该应用程序是否是只读的。
- **UpdateMode** 当应用更新时, 该属性确定如何选择记录的范围。

## 2. TSQLClientDataSet 组件主要的方法

- **CloneCursor** 与另一个数据集共享数据。
- **GetQuoteChar** 返回在产生的 SQL 语句中用来封装引用字符串的一个或多个字符。
- **LoadFromFile** 从一个文件中装载数据集的数据, 其语法如下:  

```
procedure LoadFromFile(const FileName: string = "");
```

  - **Execute** 由提供者数据集运行一个 SQL 命令。
  - **SaveToFile** 将客户数据集的数据保存到一个外部文件中, 其语法如下:  

```
procedure SaveToFile(const FileName: string = "; Format TDataPacketFormat=dfBinary);
```

    - **Close** 关闭该数据集。
    - **Edit** 使该数据进入可编辑状态。

- First 移动到第一条记录。
- Last 移动到最后一记录。
- Prior 移动到前一条记录。
- Next 移动到下一条记录。
- MoveBy 向前或向后移动多条记录。
- Post 将修改的数据提交到数据库中。
- Refresh 重新从数据库中提取数据来刷新数据集中数据的显示。

### 3. TSQLClientDataSet 组件主要的事件

其事件划分为三种类型，其中 Before 开头的事件表示发生在某种情形以前；After 开头的事件表示发生在某种情形以后；On 开头的事件表示某种情形正在出现时触发该事件。

## 6.3 建立跨平台的数据库应用程序

由于 Windows 下的 Delphi 及 Linux 下的 Kylix 都提供了 dbExpress 相关的组件，所以在 Delphi 下开发的数据库应用程序可以顺利移植到 Linux 下，只要在 Kylix 下重新对这些应用程序进行编译即可。下面通过两个实例，说明使用 dbExpress 开发数据库应用程序的具体方法。

### 6.3.1 只能浏览数据

由于 dbExpress 都是单方向的组件，它们只能使用 First 和 Next 方法导航数据，而无法使用 Prior 及 Last 等方法，也无法直接对数据进行编辑或修改，所以 TDBNavigator 组件的一些按钮将无法使用；一些数据感知控件也不能使用，比如 TDBGrid 组件，因为它会自动提交数据。下面通过一个例子说明使用 dbExpress 组件开发数据库应用程序的局限性。

该实例的实现步骤如下：

通过 File|New|CLX Application 菜单建立一个应用程序。

2. 在表单中放置组件 在表单中分别放置 dbExpress 面板上的 TSQLConnection 和 TSQLDataSet 组件各一个，默认名字分别为 SQLConnection1 和 SQLDataSet1。然后再打开 Data Access 面板，在表单中放置一个 TDataSource 组件 DataSource1。

打开 Data Controls 面板，在表单中放置 4 个 TDBEdit 组件，它们的默认名称分别为 DBEdit1、DBEdit2、DBEdit3 和 DBEdit4。

再在这些数据编辑框的下面放置一个 TDBNavigator 组件 DBNavigator1。

最后选择 Standard 面板上的标签组件，分别在 4 个数据编辑框的上面放置一个标签，名字为 Label1、Label2、Label3 和 Label4。

3. 设置组件属性 各个组件属性的设置值见表 6-4。



表 6-4 各个组件属性的设置值

组件	属性	设置值	说明
SQLConnection1	ConnectionName	IBLocal	IBLocal 数据库连接名称对应的数据库目录为： D:\Borland6\Borland Shared\Data\employee.gdb，可以在数据模块或表单中双击组件并在 Connection Settings 中设置
	DriverName	Interbase	该属性在设置 ConnectionName 后会自动设置，一般不用手工设置
	LibraryName	D:\Borland6\Delphi6\Bin\dbexpint.dll	如果没有设置该动态库的完整路径，可能会显示异常
SQLDataSet1	CommandText	select * from CUSTOMER	通过 SELECT 建立数据集
	CommandType	ctQuery	设置 CommandText 属性使用 SELECT 查询语句
	SQL Connection	SQLConnection1	通过该组件连接到数据库
DataSource1	DataSet	SQLDataSet1	设置关联的数据集
Label1	Caption	地址:	设置字段标题
	Font	蓝色/楷体/粗体/小四	
Label2	Caption	城市:	设置字段标题
	Font	蓝色/楷体/粗体/小四	
Label3	Caption	第一联系人:	设置字段标题
	Font	蓝色/楷体/粗体/小四	
Label4	Caption	第二联系人:	设置字段标题
	Font	蓝色/楷体/粗体/小四	
Edit1	DataSource	DataSource1	设置编辑框的数据源
	DataField	ADDRESS_LINE1	设置编辑框对应的字段
Edit2	DataSource	DataSource1	设置编辑框的数据源
	DataField	CITY	设置编辑框对应的字段
Edit3	DataSource	DataSource1	设置编辑框的数据源
	DataField	CONTACT_FIRST	设置编辑框对应的字段
Edit4	DataSource	DataSource1	设置编辑框的数据源
	DataField	CONTACT_LAST	设置编辑框对应的字段
DBNavigator1	DataSource	DataSource1	设置导航条的数据源

此时整个编辑界面的显示，如图 6-5 所示。

如果现在运行应用程序，单击导航条上的 Next 按钮，可以看到下一条记录。单击最后一条记录按钮，则会提示一个错误，如图 6-6 所示，提示单方向的数据源不支持该导航按钮的功能。



现在将导航条的 nbFirst 和 nbNext 设置为 True，其他按钮都设置为 False，然后重新执行应用程序，则最终显示的结果如图 6-7 所示。两个导航按钮可用于单方向的数据集。

由于该数据集是单方向的，所以无法编辑、修改及删除数据库的记录。

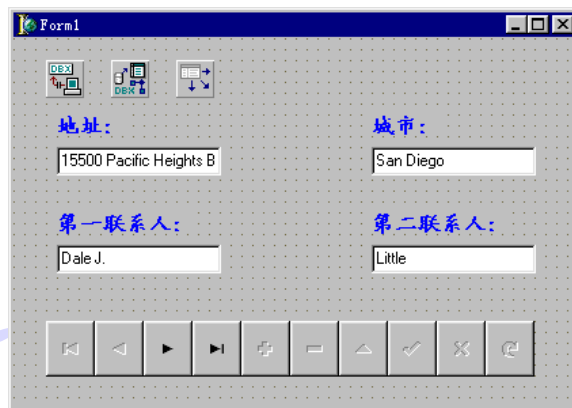


图 6-5 带有不支持的导航按钮的编辑界面

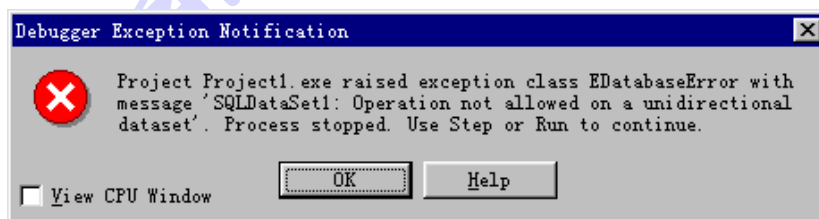


图 6-6 错误原因是使用了单向数据集不支持的功能

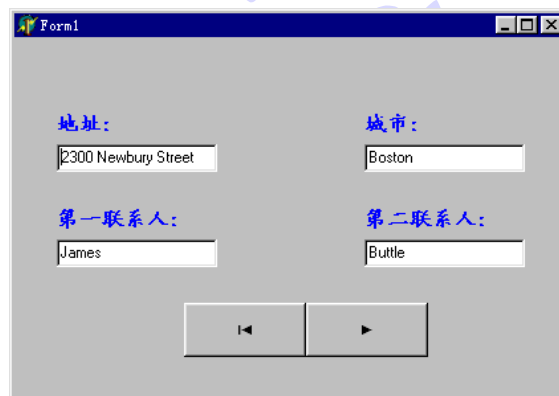


图 6-7 正确运行应用程序的显示界面

### 6.3.2 实现完整功能

由于前面建立的应用程序只能进行简单的浏览，所以不是很实用。下面通过 TSQLClientDataSet 组件建立一个可实现增、删、改、查的应用程序，由于该组件是一种客户数据集，内置了 TSQLDataSet 和 TdatasetProvider 组件，所以建立该应用程序非常简单，不可见的数据库组件只需要两个即可。

该应用程序的实现步骤如下：

1. 使用 File|New|CLX Application 新建一个跨平台的数据库应用程序。
2. 在表单上放置一个数据源组件 DataSource1，再放置一个 SQL 客户数据集组件 SQLClientDataSet1，这两个组件都是不可见组件。  
打开 Data Controls 组件面板，然后分别在表单中放置一个数据表格 DBGrid1 和导航组件 DBNavigator1。
3. 下面设置各个组件的属性，见表 6-5。

表 6-5 各个组件属性的设置值

组件	属性	设置值	说明
SQLClientDataSet1	ConnectionName	IBLocal	IBLocal 数据库连接名称对应的数据库目录为： D:\Borland6\Borland Shared\Data\employee.gdb，可以在数据模块或表单中双击组件并在 Connection Settings 中设置
	CommandText	select * from DEPARTMENT	通过 SELECT 建立数据集
	CommandType	ctQuery	设置 CommandText 属性使用 SELECT 查询语句
	Active	True	
DataSource1	DataSet	SQLClientDataSet1	设置关联的数据集
DBGrid1	DataSource	DataSource1	设置数据表格使用的数据源
DBNavigator1	DataSource	DataSource1	设置导航条的数据源

以上各个组件的属性设置完毕后，表单的显示如图 6-8 所示。

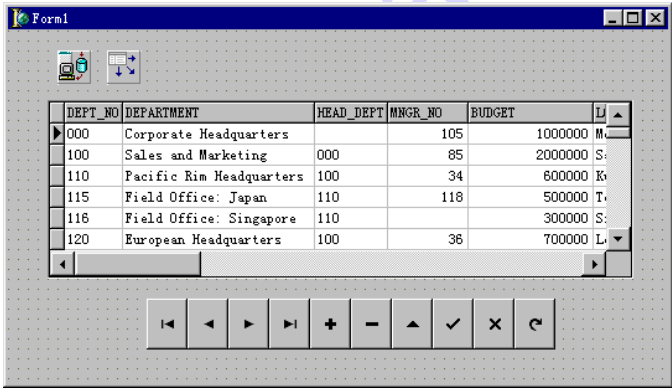


图 6-8 可以编辑及导航的表单

4. 最后运行程序，可以发现该应用程序可以实现数据增、删、查、改的功能。该程序中提取数据的流程如下（提交的数据与该过程相反）：  
InterBase 数据库→SQLClientDataSet1→DataSource1→DBGrid1→显示数据

6.3.3 改变数据单方向性

在上一节的基础上进行修改，完成应用程序的设计，具体步骤如下：

1. 先删除 SQLClientDataSet1 组件，然后打开 dbExpress 组件面板，在表单中添加一个数据库连接组件 SQLConnection1 和数据库查询组件 SQLQuery1。再打开 Data Access 组件面板，在表单中分别插入一个客户数据集 ClientDataSet1 和一个数据集提供者 DataSetProvider1。

2. 设置各个组件的属性，见表 6-6。

表 6-6 各个组件属性的设置值

组件	属性	设置值	说明
SQLConnection1	ConnectionName	IBLocal	设置连接的数据库
	DriverName	Interbase	该属性在设置 ConnectionName 后会自动设置，一般不用手工设置
	LibraryName	D:\Borland6\Delphi6\Bin\dbexpint.dll	如果没有设置该动态库的完整路径，可能会显示异常
SQLQuery1	SQL	select * from DEPARTMENT	建立数据集使用的查询语句
	SQLConnection	SQLConnection1	建立到数据库的连接
DataSetProvider1	DataSet	SQLQuery1	设置数据集提供者使用的数据集
ClientDataSet1	Active	True	激活数据集
	ProviderName	DataSetProvider1	设置使用的数据集提供者
DataSource1	DataSet	ClientDataSet1	设置关联的数据集，注意不能使用 SQLQuery1
DBGtid1	DataSource	DataSource1	设置数据表格使用的数据源
DBNavigator1	DataSource	DataSource1	设置导航条的数据源

注意：上面激活的数据集一定是 ClientDataSet1，而不要激活 SQLQuery1，因为最终数据是通过 ClientDataSet1 显示出来的。

以上各个组件的属性设置完毕后，表单的显示如图 6-9 所示。

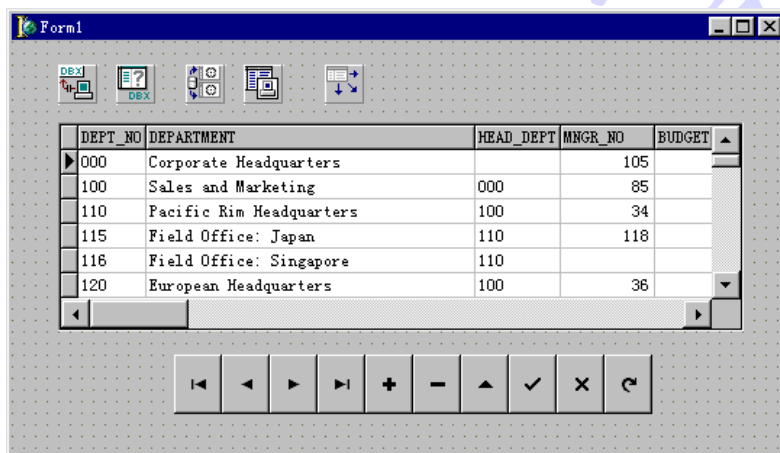


图 6-9 设计界面的最终显示

3. 运行该程序，也同样可以实现增、删、查、改的功能。该程序中提取数据的流程如下（提交的数据与该过程相反）：

InterBase 数据库→SQLConnection1→SQLQuery1→DataSetProvider1→ClientDataSet1→DataSource1→DBGrid1→显示数据

## 第 7 章 使用 Database Desktop 和 Datapump

Delphi 6 中提供了两个实用工具，分别为 Database Desktop 和 Datapump，前者用于建立、查看、修改、排列及查询不同类型的 Paradox、dBASE 及 SQL 格式的基表；后者用于将一种类型的基表传输到另一种类型的基表中。

下面分别说明这两个实用工具的作用。

### 7.1 使用 Database Desktop

使用 Database Desktop 建立及管理基表有简单、直观、方便等特点，并且还具有通用性比较强的特点，即可以使用该工具建立多种数据库的基表，为开发应用程序提供了很大的方便。将该工具与 SQL Explorer 结合，可以很好地实现其他数据库管理工具的功能。

Database Desktop 主要包括三部分内容：

- Table 部分 主要针对基表进行各种数据库操作。可通过 File|New|Table 菜单新建建立一个基表；也可通过 File|Open|Table 菜单打开一个基表来修改其结构或数据。
- QBE Query 部分 本部分是为对 SQL 语言不熟悉的用户提供的，它提供了一种直观的生成 SQL 语句的方法，用户只要在打开的基表中选择需要显示的字段，则会自动生成 SQL 语句，可通过 Query|Show SQL 菜单查看生成的 SQL 语句。可通过 File|New|QBE Query 菜单选择 SQL 查询语句中需要的字段；可通过 File|Open|QBE Query 菜单打开一个 SQL 查询语句中选择的字段并进行修改。
- SQL File 部分 直接在 SQL 编辑器内输入需要的 SQL 语句，通过菜单“SQL|RUN SQL”运行该 SQL 语句。可通过 File|New|SQL File 菜单打开一个文本编辑器并输入 SQL 语句，然后将其以 .sql 的形式保存到工作目录下；可通过 File|Open|SQL File 菜单打开一个已经建立的 SQL 语句并修改或运行它。

在使用 Database Desktop 的过程中我们会发现，如果选择的操作方式不同，则对应的菜单也不一样，所以下面分别按照共用菜单及专享菜单的方式来原因说明各个菜单的作用。

#### 7.1.1 共用菜单

在第一次打开 Database Desktop 时，可以看到有 File、Edit、Tools、Window 和 Help 等 5 个菜单项，这些菜单项分别用于文件管理、选项编辑、常用工具、窗口管理及帮助信息等方面。

##### 1. File 菜单项

该菜单项用于创建、打开及保存基表和 SQL 文件等，同时还包括了工作目录及专用目录的设置。其各个菜单功能如下：

- (1) New 菜单 该菜单会打开 3 个子菜单，分别用于建立基表、选择 SQL 语句使用

的字段及建立 SQL 语句等。其各个子菜单的作用如下：

1) QBE Query 子菜单 以图形方式显示当前选择基表的各个字段，然后可以从中选择需要使用的字段，则最终生成的 SQL 查询语句只会查询这些字段的内容，可以通过专用菜单 Query|Show SQL 查看生成的 SQL 语句。选择该菜单后会弹出一个文件选择窗口，可以从相应目录中选择一个需要进行数据操作的基表，然后进行确认，则会弹出该基表名及其所有字段名的显示窗口，可以从中选择一个或多个字段，要选择一个字段，可在该字段左边的单选框内单击鼠标，此时会出现一个对号。比如我们使用 Delphi 6 提供的 country.db 数据库基本表，则选择 Name 和 Capital 字段后界面的显示如图 7-1 所示。单击 Query|Show SQL 菜单可看到生成的 SQL 语句，如图 7-2 所示。单击 Query|RUN SQL 菜单可检验 SQL 语句执行的结果，如图 7-3 所示。

退出该菜单时，会弹出一个对话框，询问是否保存已经建立的图形化界面，如果要保存，可以输入一个文件名，比如为 country，其文件扩展名是.qbe。该文件默认保存在工作目录下。

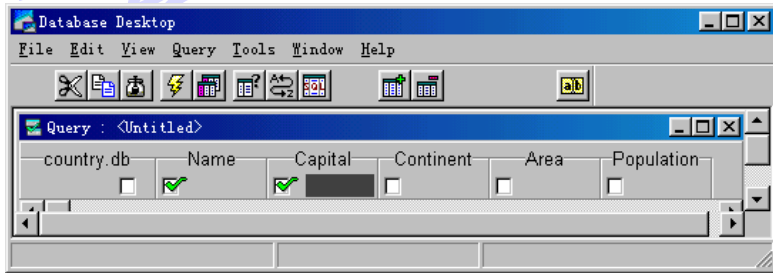


图 7-1 以图形化的方式建立 SQL 查询语句

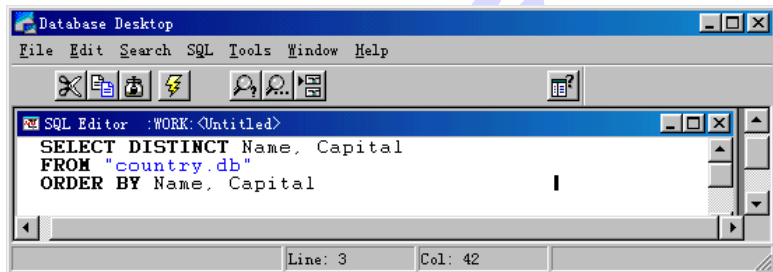


图 7-2 对应选择字段形成的 SQL 查询语句



图 7-3 运行 SQL 语句后查询的结果



2) SQL File 子菜单 该菜单会打开一个 SQL 编辑器, 在该编辑器内可直接输入相应的 SQL 语句。注意, SQL 语句可以是一般 SQL 数据库服务器使用的标准 SQL 语句, 也可以是 Delphi 定义的本地 SQL 语句, 二者的差别不是很大。比如 dBase、Paradox 等数据库一般使用本地 SQL 语句, 而 Oracle、SQL Server 等使用标准 SQL 语句。不过一般使用标准 SQL 语句即可。编辑 SQL 语句的窗口如图 7-4 所示。在关闭该编辑框时, 系统会询问是否需要保存该 SQL 语句, 如果选择保存, 则会以 .sql 扩展名保存该文件。该文件也会保存到工作目录下。

**注意:** 标准 SQL 与本地 SQL 语句的一个重要区别是: 本地 SQL 语句中的基表要加单引号 “'” 或双引号 “””, 同时要使用基表所在的完整路径, 而标准 SQL 语句不需要引号。

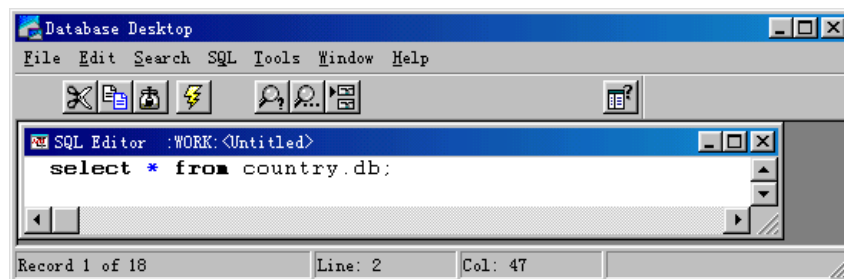


图 7-4 SQL 语句编辑框

3) Table 子菜单 用于建立一个新的基表。该菜单打开的对话框如图 7-5 所示。可以从 Table type 下拉框中选择需要建立的基表类型, 默认是建立 Paradox7 类型的基表, 然后单击 OK 按钮, 则会打开基表结构建立面板, 建立了 4 个字段的面板如图 7-6 所示。

建立基表字段的方法如下: 将光标移动到 Field Name 一栏, 然后输入第一个字段的名字, 该名字最好使用英文名字, 这样比较容易处理; 按键盘上的 Tab 键或者使用鼠标, 将光标移动到 Type 一栏, 然后单击鼠标右键, 并从弹出菜单中选择合适的字段类型, Paradox 7 使用的字段类型的作用见表 7-1; 再将光标移动到 Size 一栏, 如果是字符类型, 则输入该字段需要的最大长度, 然后输入一个数字, 表示字段使用的字节数, 应注意一个汉字使用两个字节, 另外该长度应设置得稍微大一些, 以保证将来能够容纳下最长的字段内容, 如果是数字等类型, 就不要在该栏输入长度; 如果该字段是关键字段, 则可以在 Key 一栏双击鼠标添加一个 “\*”, 如果不是关键字段, 则保留该栏为空。

按 Tab 键, 将光标移动到第二行, 然后按照上面的方法再继续输入下一个字段的内容即可。

在建立基表时, 最好提前将基表的结构设计好, 这样就会提高建立基表的效率。

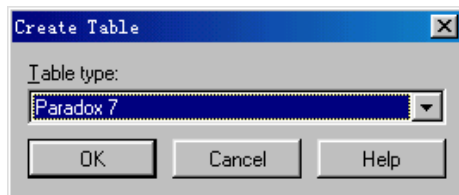


图 7-5 选择建立基表的类型

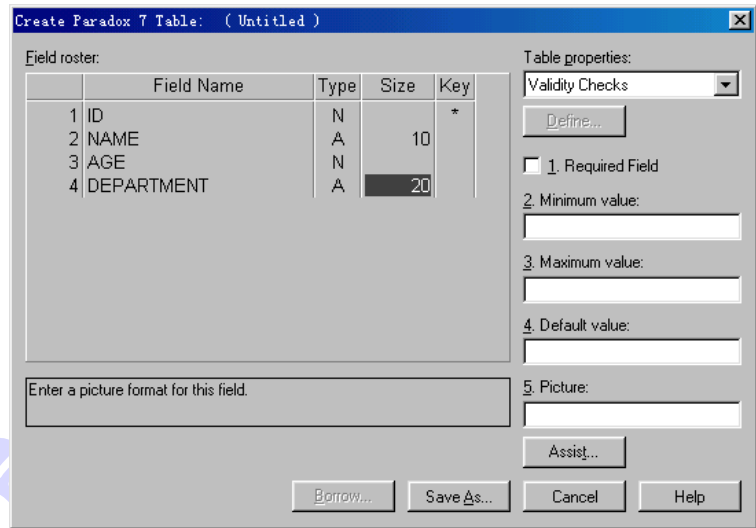


图 7-6 建立基表结构的面板

表 7-1 Paradox 常用的字段类型及作用

符号	名称	大小	说明
A	Alpha	2~255	字符类型
N	Number		实数格式，只能输入数字
\$	Money		货币显示方式，格式如¥12.00，只能输入数字
S	Short		范围为-32, 768 到 32, 767 的整数，只能输入数字
I	Long Integer		超过 Short 范围的整数，只能输入数字
#	BCD	0 ~ 32**	一种二十进制编码，需要设置小数点后位数
D	Date		日期格式如 1989-1-21
T	Time		时间格式如 12:35:01，时、分、秒要用冒号隔开
@	Timestamp		包括时间及日期，格式如 12:01:35, 1990-10-21
M	Memo	1 ~ 240**	输入 Alpha 无法容纳的字符串及文本等
F	Formatted Memo	0 ~ 240**	设置输入文本的格式，比如字体格式、颜色等
G	Graphic	0 ~ 240**	显示.BMP、.PCX、.TIF、.GIF、.EPS 格式图像
O	Ole	0 ~ 240**	同其他程序建立 Ole 连接
L	Logical		只有 True 及 False 两个值
+	±Autoincrement		自动递增 1，主要用于序号等
B	Binary	0 ~ 32**	用于声音文件等
Y	Byte	1~255	用于条形码等，实际上很少用到

建立基表结构的面板中（见图 7-6）的各个选项的意义如下：

- Field roster 该部分用于建立基表的基本结构。
- The field type is missing. 该方框中的内容是对建立基表时存在的问题的提示。

- **Table properties** 用于设置基表的一些属性，比如有效性检查、第二个索引、口令安全等。
- **Define 按钮** 单击该按钮可以打开一个面板来定义基表的一些属性。
- **Modify 按钮** 该按钮只有在 Define 按钮可用时才显示出来，用于修改已经定义的基表属性。
- **Required Field** 该复选框用于限制一个字段是否不能为空，一般关键字段都选中该选项。
- **Maximum value 和 Minimum value** 这两个选项用于设置当前字段允许输入的最大值和最小值。
- **Default value** 设置当前字段没有输入值时使用的默认值。
- **Picture** 如果一个字段需要使用图像，则可以单击下面的 Assist 按钮为该字段设置一个关联的例子图像。
- **Borrow 按钮** 用来拷贝其他基表的结构。
- **Save as** 保存建立完成的基表。
- **Cancel** 取消当前建立的基表。
- **Help** 获得当前面板的帮助。

(2) **Open 菜单** 该菜单用于打开已经建立的基表、SQL 文件或 QBE 查询等，然后可以编辑或修改这些内容。

(3) **Close 菜单** 关闭新建立或已打开的内容，该菜单只有在打开其他的内容后才可用。

(4) **Save 菜单** 保存当前新建立或打开的基表结构、SQL 文件或 QBE 查询等。

(5) **Save As 菜单** 以其他名字保存当前编辑的内容。

(6) **Working Directory 菜单** 建立 Database Desktop 的工作目录，该菜单会打开一个对话框，如图 7-7 所示，可以在 Working Directory 编辑框中输入一个已经建立的目录作为工作目录，这样在 Database Desktop 建立的基表、SQL 文件及 QBE 查询等默认会保存在该目录下。Aliases 编辑框中可以输入一个别名，也可以保持为空，这样会自动使用默认的别名：WORK。

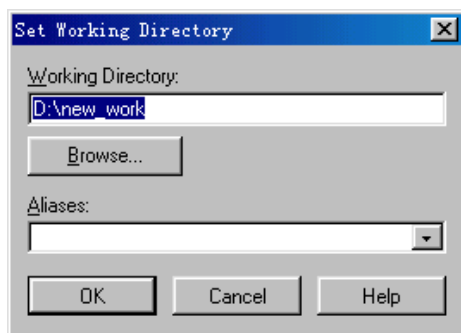


图 7-7 设置 Database Desktop 的工作目录

(7) **Private Directory 菜单** 用于建立一个专用的私有目录来保存建立的临时基表，从而可以避免与其他网络用户的临时基表混淆。可以使用系统设置的默认私有目录，也可

以重新建立一个私有目录，建立私有目录的对话框如图 7-8 所示。

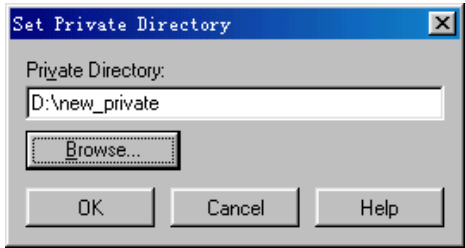


图 7-8 设置自己的私有目录

(8) Exit 菜单 退出 Database Desktop 应用程序。

2. Edit 菜单项 该菜单项用于实现一些编辑功能，第一次打开 Database DeskTop 时，只有一个菜单：

(1) Preferences 菜单 该菜单用于设置系统的一些属性，比如使用的字体、是否显示工具条等，它共有两个标签页：General 标签页用于设置设置系统字体；ToolBars 标签页决定是否显示标准工具条及全局工具条。其中，选择 Standard 复选项会打开通用工具条，选择 Global 复选项会显示打开 Table、QBE Query、SQL File 的快捷工具，相当于 File|Open 菜单。

3. Tools 菜单项 该菜单项提供了一些实用工具，各个菜单的功能如下：

(1) Alias Manager 菜单 该菜单主要用于建立或删除数据库别名。数据库的类型不同，则打开的别名管理面板会不同，Delphi6 提供的 DBDEMOS 数据库别名对应的别名管理面板如图 7-9 所示。

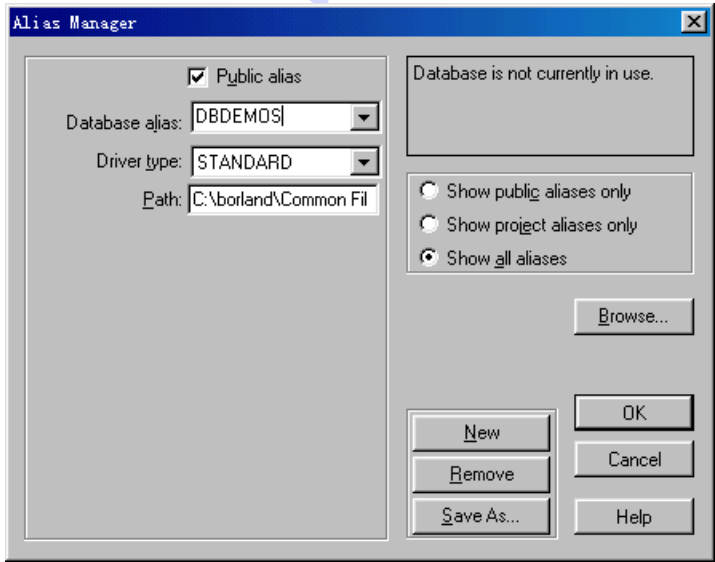


图 7-9 数据库别名管理面板

注意：由于只有使用 BDE 开发的数据库才使用数据库别名，所以该菜单不能用于 ADO、dbExpress、Interbase 等面板的数据库组件中。

(2) Utilities 菜单 该菜单包括了多个子菜单，主要用于管理数据库的基表。各个子

菜单的功能如下：

1) Add 子菜单 用于将一个基表的记录追加到另一个有相同数据类型的基表中。首先应选择基表源，接下来需要选择目标基表。可采用追加（Append）、更新（Update）或追加和更新（Append&update）等方式在两个基表间拷贝数据，选择目标基表及数据追加方式的面板如图 7-10 所示。

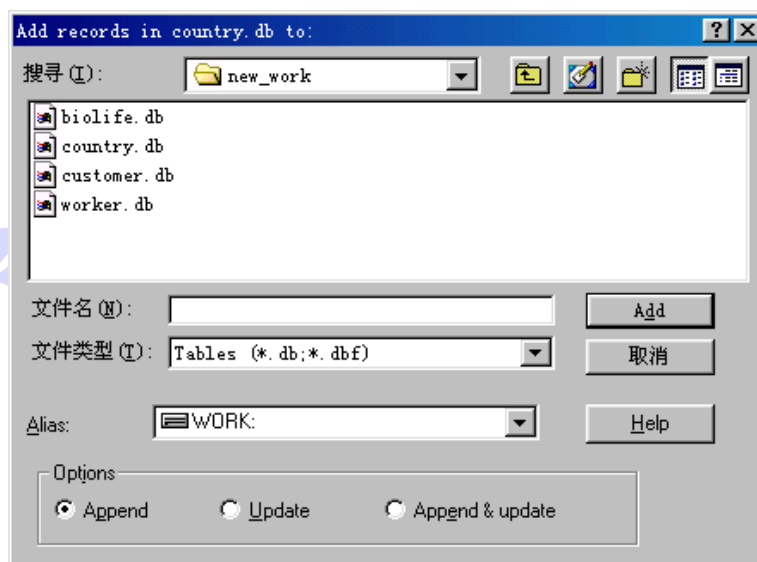


图 7-10 选择目标基表及基表间数据的拷贝方式

2) Copy 子菜单 将文件从一个位置拷贝到另一个位置，可以是 Tables、Queries、SQL 或文本文件。如果拷贝的是一个基表，则拷贝该基表时也会拷贝与其相关的一些文件，比如索引文件等，同时如果为拷贝的文件重新命名，则其关联文件也应相应修改名字。

3) Delete 子菜单 用于删除一个文件，可以是 Tables、Queries、SQL 或文本文件。如果删除的是一个基表，则也会删除与该基表相关的一些文件，比如索引文件等。

4) Empty 子菜单 删除一个基表中所有的数据。

5) Info Structure 子菜单 显示基表的定义。

6) Rename 子菜单 重新命名一个文件。如果修改了一个基表的名字，则也会修改与该基表相关的一些文件名，比如索引文件等。

7) Sort 子菜单 将基表中的数据排序。可以使用原基表保存排序的结果，也可以使用一个新的基表保存排序的结果，该新的基表同时也拷贝了原来基表的结构和数据。该菜单打开的面板如图 7-11 所示。

该面板中各个选项的意义如下：

- Sorted table 该部分的单选按钮和复选按钮选项用于决定如何为基表的数据排序。各个选项的作用如下：

◇ Same Table 选择该选项，则数据排序后仍保存在原来的基表中。实际上为该基表建立了一些索引。

◇ New Table 该选项表示使用一个新的基表保存原基表排序后的数据及结构。

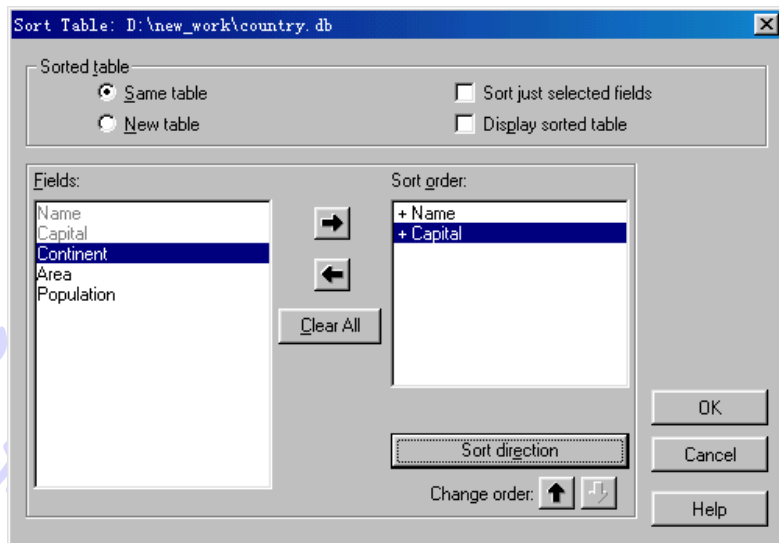


图 7-11 将基表中的数据排序

◇ **Sort Just Selected Fields** 选择该选项后，则只会对 **Sort order** 选项中列出的字段数据进行排序。

◇ **Display sorted table** 选择该选项后，可以立即看到数据排序后的结果。

- **Fields** 该列表框中列出了当前基表中的所有字段，可以选择一个或多个字段，使用右边的箭头将这些字段移动到 **Sort order** 中，以便按照这些字段对数据进行排列。
- **Sort order** 该列表框中列出的字段就是该基表排序的字段，先按照前面的字段排序，然后再按照后面的字段排序。可以使用下面 **Change order** 右边的箭头按钮来改变字段排列的先后顺序。如果一个字段左边有一个加号“+”，表示按照该字段的升序排列；如果一个字段左边有一个减号“-”，表示按照该字段的降序排列。用鼠标双击一个字段，则可以在加号“+”和减号“-”之间切换。也可以单击下面的 **Sort direction** 按钮来改变一个字段的排序方式。
- **Clear All** 该按钮可以清除 **Sort order** 列表中所有的排序字段。

8) **Restructure** 子菜单 用于修改一个基表的结构。也可以使用 **Table/Restructure** 菜单修改一个基表的结构。

9) **Subtract** 子菜单 该菜单用于删除第二个基表中与第一个基表中相同的那些记录。

(3) **Passwords** 菜单 为基表设置一个口令，以便保护基表的数据。

4. **Windows** 菜单项 控制多个窗口的显示方式。其包含的子菜单功能如下：

- (1) **Cascade** 菜单 自左向右依次排列所有的窗口，窗口重叠。
- (2) **Tile Top to Bottom** 菜单 自上向下依次排列所有的窗口，窗口不重叠。
- (3) **Tile Side-by-Side** 菜单 自左向右依次排列打开的窗口，窗口不重叠。
- (4) **Arrange Icons** 菜单 重新排列所有的图标。
- (5) **Close All** 菜单 关闭已经打开的所有窗口。

5. **Help** 菜单项 提供 Database Desktop 的帮助信息，有两个子菜单：

- (1) **User's Guide Topic** 菜单 相当于 Database Desktop 用户指南，可以通过内容列表



或索引来查找需要的帮助信息。

(2) About Database Desktop 菜单 显示当前使用的 Database Desktop 版本等信息。

### 7.1.2 专用菜单

通过 File|Open 菜单打开的文件不同, 则 Database Desktop 的系统菜单也会做相应的变化, 它们分别增加了一些新的菜单项, 同时 Edit 菜单的子菜单也增加了许多, 但是 File、Tools、Window 和 Help 等菜单项包括的菜单基本上没有变化, 所以下面根据打开的文件不同, 分别对应菜单的不同作用。

#### 1. 打开 QBE Query 时使用的专用菜单项

(1) Edit 菜单 该菜单项主要实现当前打开文件的一些编辑功能。其各个子菜单的作用如下:

- 1) Add Table 菜单 将一些基表添加到当前打开的 QBE Query 中。
- 2) Remove Table 菜单 用于删除当前 QBE Query 使用的基表。
- 3) Cut 菜单 剪切选择内容到剪贴板上。
- 4) Copy 菜单 拷贝选择内容到剪贴板上。
- 5) Paste 菜单 将剪贴板上的内容粘贴到光标位置。
- 6) Paste Link 菜单 通过 DDE 实现数据的连接备份。
- 7) Delete 菜单 删除当前选择的内容。

(2) View 菜单项 该菜单项实现一些数据浏览功能, 各个子菜单的作用如下:

- 1) Field View 菜单 用于浏览一个字段的的数据。
- 2) Tile Tables 菜单 该菜单将当前 QBE Query 中显示基表字段的窗口按照从上向下的顺序排列。
- 3) Cascade Tables 菜单 该菜单将当前 QBE Query 中显示基表字段的窗口按照从左上角向右下角的顺序排列。

(3) Query 菜单项 该菜单项用于实现 QBE Query 的一些特定功能, 其各个子菜单的作用如下:

- 1) Run Query 菜单 运行 QBE Query 中建立的查询并显示查询结果。
- 2) Wait for DDE 菜单 该菜单使查询语句中使用的 DDE 数据随 DDE 中数据的改变而变化。
- 3) Show SQL 菜单 显示 QBE Query 对应的 SQL 语句。
- 4) Ignore Changes 菜单 该菜单在 Database Desktop 运行一个查询时, 允许其他用户对基表中的数据进行修改, 防止 Database Desktop 重新启动该查询。
- 5) Lock Tables 菜单 自己使用基表时可锁定这些基表以防止其他人同时修改这些基表的数据。
- 6) Restart on Changes 菜单 如果修改了一个基表, 使用该菜单可使 Database Desktop 自动重新执行查询。
- 7) Properties 菜单 用于设置 QBE Query 的一些属性, 打开的面板包括 4 个标签页, 各个标签页的作用如下:
  - Answer 标签页 用于设置基表的类型及基表的名字。

- QBE 标签页 用于设置查询的执行方式。
- Sort 标签页 用于设置基表中数据的排列顺序。
- Structure 标签页 用于显示查询语句中使用的字段。

其中，Answer 标签页的显示如图 7-12 所示。

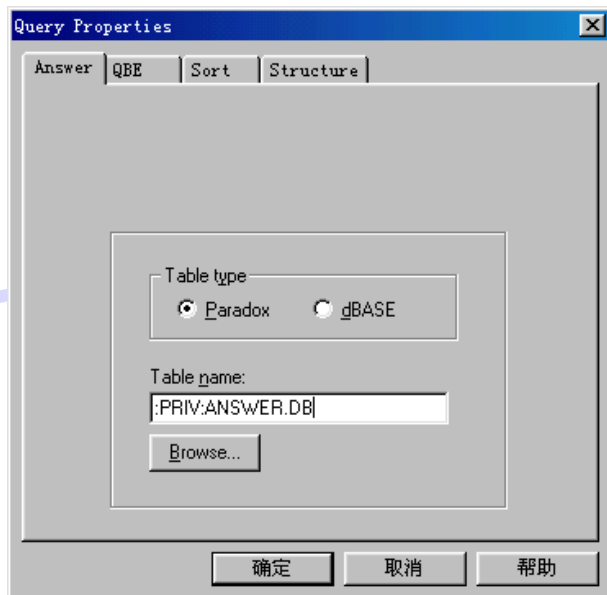


图 7-12 QBE 标签页的显示

## 2. 打开 SQL File 时使用的专用菜单

SQL File 有三个专用菜单项，分别是 Edit、Search 及 SQL，其功能如下：

(1) Edit 菜单项 该菜单项为 SQL 编辑器提供一些编辑功能。

1) Cut、Copy、Paste 和 Delete 菜单 这些菜单与 QBE Query 中对应的菜单作用相同。

2) Undo 取消已经进行的操作，可取消多步操作。

3) Copy to 菜单 拷贝选择的内容到一个扩展名为.txt、.qbe、.sql 的文件中。

4) Paste From 菜单 将另一个文本文件中选择的内容拷贝到当前文本光标指示的位置。

5) Select All 菜单 选择所有内容。

6) Editor Preferences 设置 SQL 文件编辑器的一些选项，包括三个标签页：

- Editor 标签页 设置一些编辑特性，比如是否自动缩进、是否可以成块覆盖内容等。
- Display 标签页 设置一些显示特性，比如设置键击的映射、使用的字体、状态条提示等一些选项。
- Colors 标签页 设置 SQL 编辑器中不同类型的语句或内容使用的默认颜色。

其中，Editor 标签页的显示如图 7-13 所示。

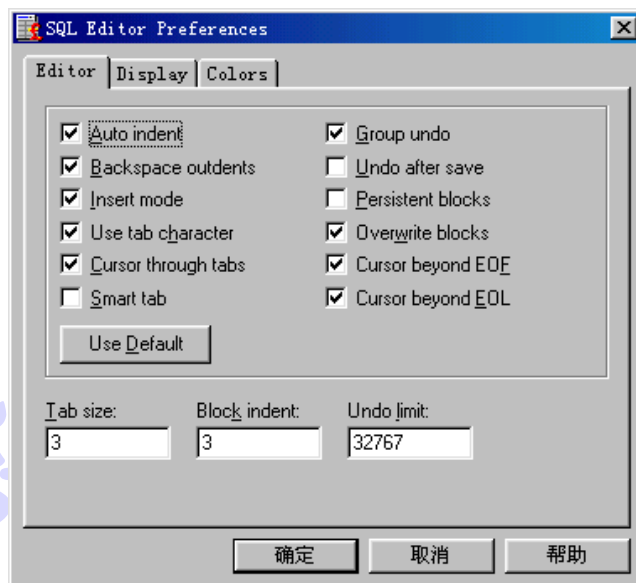


图 7-13 SQL 编辑器选项的 Editor 标签页

(2) Search 菜单项 为 SQL 编辑器提供查找及替换功能，它包括的子菜单如下：

1) Find 菜单 用于查找符合条件的内容。打开的面板如图 7-14 所示，其各个选项及按钮的作用如下：

- Search for 设置需要被替换的内容。
- Case sensitive 选择该复选框，表示查找的内容大小写敏感。
- Backwards 选择该复选框，表示由后向前查找。
- Whole words only 选择该复选框，表示查找只匹配完整的单词或汉字。
- Advanced Pattern Match 选择该复选框，表示可以使用 Database Desktop 的扩展通配符搜索符合条件的内容。
- Find 按钮 执行设置内容的搜索。

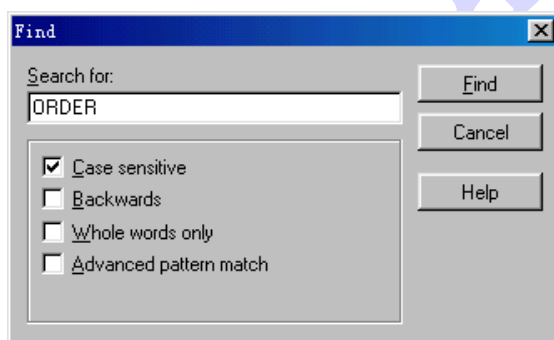


图 7-14 搜索对话框

2) Find Next 菜单 按照已经执行的搜索，继续查找下一个匹配内容。

3) Replace 菜单 先查找符合条件的内容，然后使用新的内容替换这些内容。其打开的面板中多个选项都与 Find 菜单打开的面板中的选项相似，增加的一些选项及按钮如下：

- Replace with 设置替换的新内容。

- Replace All 单击按钮，则使用新内容替换所有符合条件的内容。
- 4) Replace Next 菜单 替换下一个符合条件的内容。
- 2. SQL 菜单项 该菜单项主要用于运行 SQL 语句及设置 SQL 语句的一些属性等，包括以下子菜单：
  - 1) RUN SQL 菜单 运行 SQL 编辑器中的 SQL 语句。
  - 2) Select Alias 菜单 用于选择一个远程数据库的别名，以便向其发送 SQL 查询语句。
  - 3) Properties 菜单 该菜单用于设置 SQL 语句的一些属性。包括两个标签页：
    - Answer 设置查询答复的类型。
    - SQL 设置是在本地还是远程运行一个查询，同时还设置了一些辅助基表选项及约束。该标签页如图 7-15 所示。

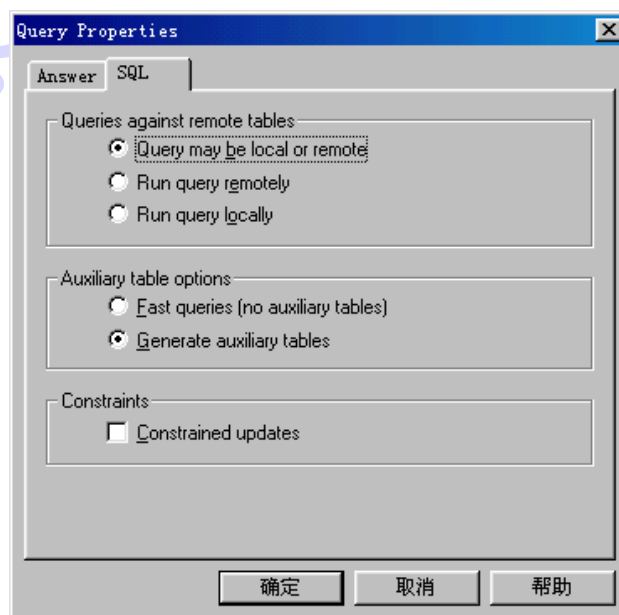


图 7-15 查询属性面板的 SQL 标签页

3. 打开基表时使用的专用菜单
  - (1) Edit 菜单项 该菜单项用于实现基表的一些编辑功能，可参考 QBE Query 及 SQL 文件使用的菜单。
  - (2) View 菜单项 该菜单项用于浏览数据库数据，该菜单项下的子菜单作用如下：
    - 1) Field View 菜单 该菜单可用于浏览单个字段的数据，可以查看一些数据精确度很高的值。
    - 2) Persistent Field View 菜单 该菜单与 Field View 菜单作用相似，只是可提供持久的字段浏览功能。
  - (3) Table 菜单项 该菜单项主要用于基表的管理等。包含的子菜单作用如下：
    - 1) Edit Data 菜单 该菜单用于编辑数据。
    - 2) View Data 菜单 该菜单用于浏览基表的数据。
    - 3) Info Structure 菜单 显示基表的结构信息，与 Tools|Utilities| Info Structure 菜单作用

相同。

4) Restructure 菜单 用于修改基表的结构。

5) Notify on 菜单 该菜单在 DDE 数据传输中控制传送到客户端的数据。

6) Strict Translation 菜单 该菜单用于限制输入的字符集，以便符合 DOS 或 Windows 系统的标准。

7) Table View Properties 菜单 该菜单用于保存基表的属性，其子菜单的作用如下：

- Save 子菜单 保存基表中修改的属性。Paradox 基表将修改的属性保存到.TV 文件中，dBASE 基表使用.TVF 文件。
- Restore 子菜单 恢复上次保存基表属性后已改变的属性。
- Delete 子菜单 删除以前保存的基表属性文件。

(4) Record 菜单项 该菜单项可实现对基表记录的操作，各个菜单的作用如下：

1) Next 菜单 移动到下一条记录。

2) Previous 菜单 移动到前一条记录。

3) Next Set 菜单 移动到下一页的记录。

4) Previous Set 菜单 移动到前一页的记录。

5) First 菜单 移动到第一条记录。

6) Last 菜单 移动到最后一条记录。

7) Insert 菜单 在光标位置插入一条新记录。

8) Delete 菜单 删除当前选择的记录。

9) Lock 菜单 锁定当前选择的记录，离开该记录后会自动解除对该记录的锁定。

10) Post/Keep Locked 菜单 保持对一条记录的锁定状态。

11) Lookup Help 菜单 该菜单用于从另一个基表的一个字段中查找数据完成当前字段的输入。

12) Move Help 菜单 该菜单用于将从属表的一条记录移动到主表中一条新记录上。

### 7.1.3 SQL 语句、QBE Query 及 Table 的比较

#### 1. 使用 SQL 语句查询数据

使用 File|New|SQL File 菜单打开 SQL 文件编辑器，然后在该编辑器中输入一些内容，如图 7-16 所示。该 SQL 语句只查询 Country.db 中的三个字段，同时通过 Name 字段限制了查询的内容。可以使用 SQL|Run SQL 菜单执行该 SQL 语句来查看查询的结果。

选择菜单 File|Save 保存该文件，扩展名为.sql，比如保存为 aa.sql，该 SQL 文件也可以应用到 Oracle 提供的 SQL\*PLUS 等应用程序中。

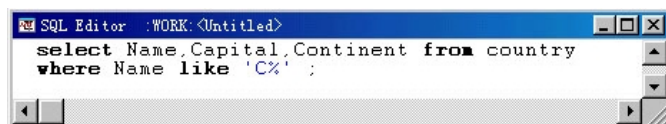


图 7-16 在 SQL 文件编辑器输入 SQL 查询语句

#### 2. 使用 QBE Query 查询数据

使用 File|New|QBE Query 菜单选择 country.db 基表，然后打开 QBE Query 设置面板，

在该面板中选择前面的 3 个字段，如图 7-17 所示。选择 Query|Show SQL 菜单显示其对应的 SQL 语句，然后选择 Query|Run Query 菜单执行该 SQL 语句，可以看到查询出基表中所有的记录，但是只显示了选择的 3 个字段的数据，而没有显示其他字段的数据。

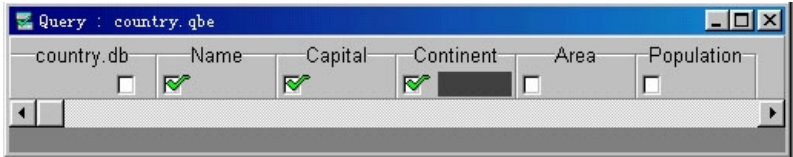


图 7-17 使用 QBE Query 面板建立 SQL 查询语句

3. 使用 Table 显示基表数据

选择 File|Open|Table 菜单后，从文件选择框中选择 country.db，然后按“打开”按钮，则会打开该基表，并显示出所有字段和所有记录的数据，如图 7-18 所示。可以使用 Table 菜单项中的菜单对这些数据进行修改或浏览。

Capital	Continent	Area	Population
Buenos Aires	South America	2,777,815.00	32,300,003.00
La Paz	South America	1,098,575.00	7,300,000.00
Brasilia	South America	8,511,196.00	*****
Ottawa	North America	9,976,147.00	26,500,000.00
Santiago	South America	756,943.00	13,200,000.00
Bagota	South America	1,138,907.00	33,000,000.00
Havana	North America	114,524.00	10,600,000.00
Quito	South America	455,502.00	10,600,000.00
San Salvador	North America	20,865.00	5,300,000.00

图 7-18 显示基表的所有数据

4. SQL 语句、QBE Query 及 Table 的比较

由上面几种方法查询数据的结果可以看出：直接编写 SQL 语句比较灵活，可以在 SELECT 语句中选择查询哪些字段，也可以通过 Where 语句限制查询的记录数量，另外，还可以执行 UPDATA、DELETE 或其他 SQL 语句；QBE Query 建立查询的 SQL 语句比较直观，可以很方便地选择显示的字段，但是不能使用 Where 语句限制查询的记录数量；Table 则会显示出基表的所有字段和数据，无法只显示部分字段或记录的数据。

但是，直接编写的 SQL 语句和通过 QBE Query 建立的 SQL 语句都可以通过运行，可以与 Table 打开的基表数据使用相同的界面。它们都可以实现对数据的编辑或浏览等功能。

7.2 在不同数据库间移动基表

Delphi 6 中提供了一个非常实用的工具 Data Pump，可以实现本地数据库或 SQL 数据库间的基表及其数据的转移，即从源数据库的基表中读取数据，然后移动到目标数据库的基表中，在移动过程中，源基表的数据类型会根据目标数据库的数据类型进行相应的变化。

注意：Data Pump 只适用于使用 BDE 的数据库。



### 7.2.1 数据库间基表移动的步骤

使用 Data Pump 实现数据库基表之间移动的步骤如下：

1. 使用 BDE 管理器分别为源和目标数据库建立一个别名。
2. 在 Data Pump 中，选择源和目标数据库别名，源数据库可以是一个数据库别名或数据库的目录。但是，SQL Server 数据库只能使用一个别名，并且需要注册。
3. 从源数据库中选择需要移动的基表。
4. 查看最初的报告信息，以便决定将数据移动到目标数据库中时，这些数据如何显示。
5. 如果目标数据库中不支持一些数据类型、索引或数据的完整性等，则修改这些内容。
6. 开始数据的移动。
7. 查看最终的状态报告信息，以便决定数据对象移动的顺序、移动哪些数据对象及这些数据对象如何在目标数据库中等内容。然后可以直接在目标数据库中修改或更新这些数据。

注意：一些数据库还有特殊的限制，具体限制如下：

(1) Microsoft SQL Server 如果目标数据库是 Microsoft SQL Server (即 MSSQL)，则应在 BDE 管理器中将 MAX QUERY TIME 和 TIMEOUT 参数设置为 20 (其默认值为 300)，否则在移动数据时会遇到问题。

(2) 如果目标数据库是 Paradox 基表，则移动数据时不要同时移动源数据库的各种索引，因为 Paradox 在建立第二条索引以前，必须有一个主索引。

### 7.2.2 数据库间基表移动的实例

下面通过一个实例，说明如何通过 Data Pump 实现数据库间基表数据的移动。该实例使用的源数据库是一个 Interbase 数据库，其路径是 D:\InterBase\examples\database\employee.gdb，数据库别名是 INTRBASE1。使用的目标数据库是 Delphi 提供的数据库，其数据库别名是 DBDEMOS，路径是 C:\borland\Common Files\Borland Shared\Data。

该实例的实现步骤如下：

1. 从 Delphi 6 的程序组中选择 Datapump，则会打开 Data Pump 向导的第一屏，该屏用于选择一个源数据库。有两个选项：

- Select by alias name 选择该选项，则按照数据库的别名选择数据库。
- Select by directory 选择该选项，则会打开一个文件管理器，可以通过文件的目录来查找使用的数据库。

该实例使用第一个选项，从右边的数据库别名列表中选择 INTRBASE1。此时，该屏的显示如图 7-19 所示。然后单击 Next 按钮，此时会弹出一个对话框，要求输入用户名及口令，可以输入 sysdba 的用户名和 masterkey 的口令，确认后打开下一屏。

2. 第二屏用于选择目标数据库。可以从目标数据库别名列表中选择 DBDEMOS，此时该屏的显示如图 7-20 所示。

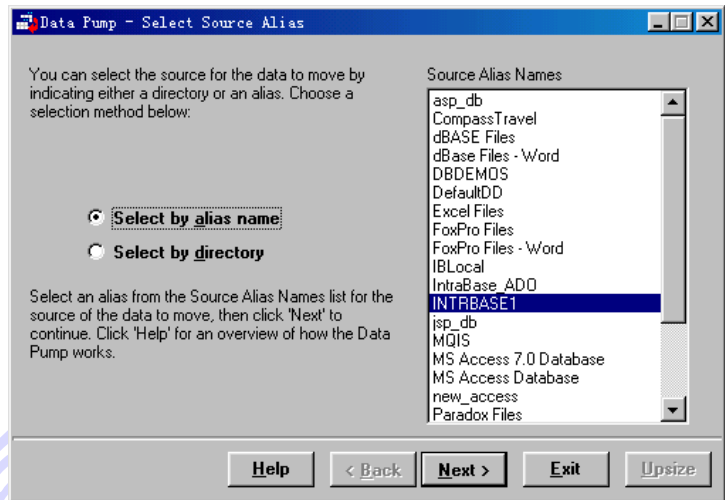


图 7-19 选择源数据库的别名

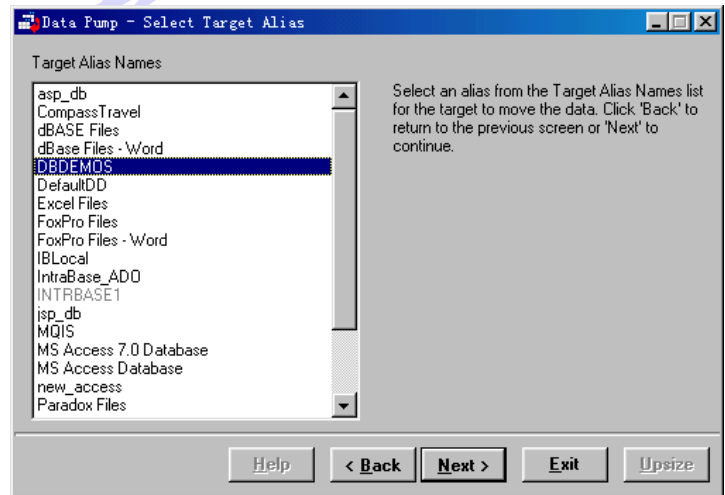


图 7-20 选择目标数据库别名

(3) 单击 Next 按钮，可以打开第三屏，该屏左边的列表框中会列出源数据库 employee.gdb 中所有的基表，可以选择一个或多个基表，然后使用按钮“>”可以将其移动到左边的列表中。单击“>>”按钮，会将源数据库中所有的基表都移动到右边的列表框中。如果要取消对一个基表的移动，可以在左边的列表框中选择该基表，然后单击“<”按钮重新将其移回左边的列表框中。单击“<<”按钮，则会将已经选择的所有基表都重新移回左边的列表框中。

本实例选择下面的两个基表，此时该面板的显示如图 7-21 所示。

(4) 单击 Next 按钮，则会打开第四屏，该屏列出了要移动的基表的一些信息，比如字段、索引、完整性约束等是否改变及是否通过了验证。该屏的显示如图 7-22 所示。此时如果一个单元格中显示“Unchanged”，则表示该基表的字段或索引不需要改变。

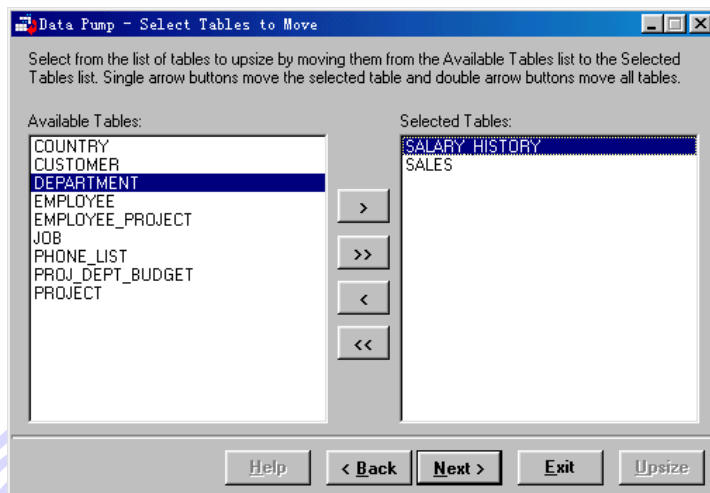


图 7-21 选择要移动的基表

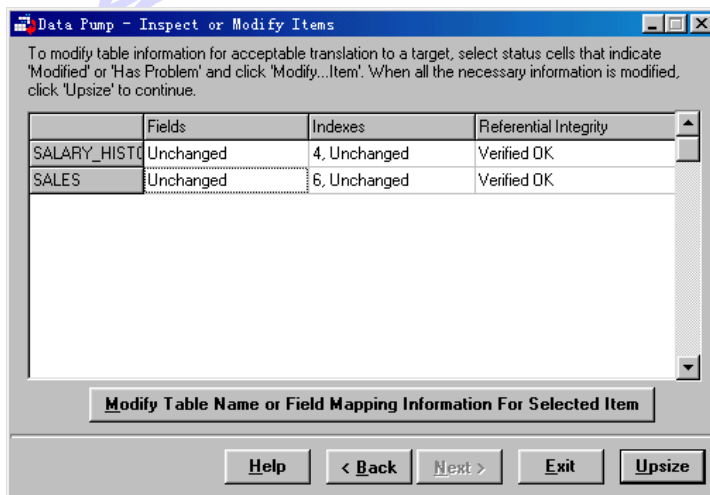


图 7-22 显示基表移动的信息

(5) 如果某个单元格显示“Modified”和“Has Problem”等信息，则应单击下面的“Modify Table...Item”按钮来修改相应的内容。比如，如果选择 SALES 基表的 Fields 一栏对应的单元格，然后单击下面的按钮，则打开的面板如图 7-23 所示。该面板的右边列表框中列出了源基表的所有字段，右边两栏则列出了源基表的字段与目标基表字段在字段名称、数据类型、是否为空值等方面的对应关系。变暗的内容表示不需要设置。

需要修改的内容都修改完毕后，单击 Next 按钮会返回到上一屏。

(6) 如果所有内容都设置正确后，可以单击 Upsize 按钮，会进行基表数据的转移，转移完成后，会显示出一些统计信息，如图 7-24 所示。可以单击下面的“Write...file”按钮，将这些信息保存到一个 Paradox 基表中。

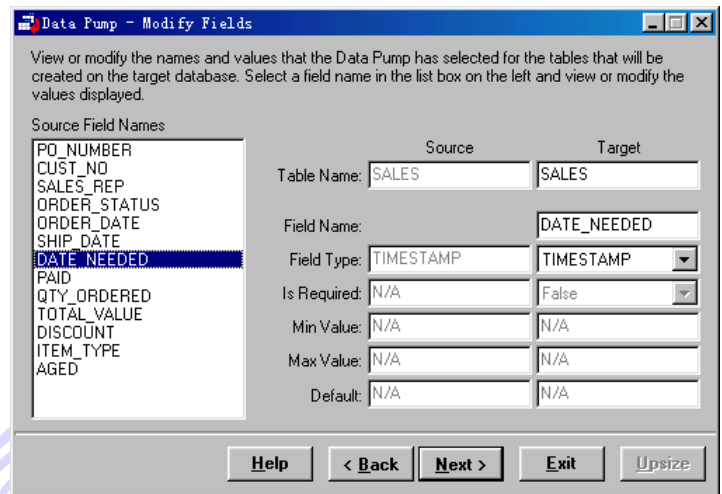


图 7-23 修改目标字段的面板

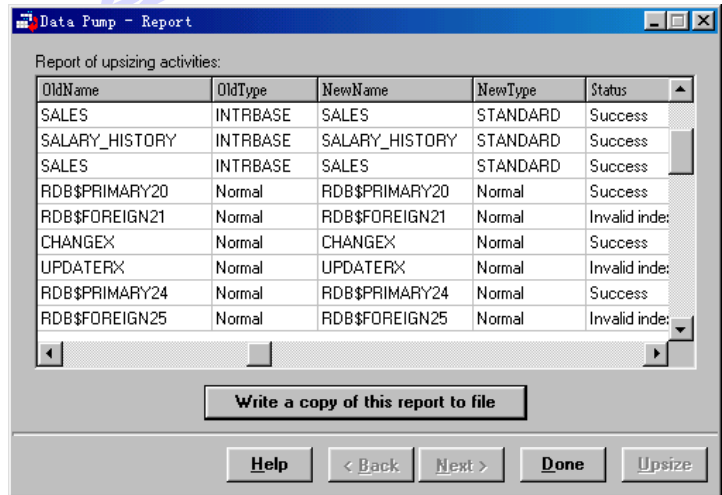


图 7-24 基表数据移动后的统计信息

(7) 最后单击 Done 按钮，可关闭 Data Pump 向导。这样，源数据库中选择的基表就彻底移动到了目标数据库中。

## 第 8 章 数据库综合实例

前面几章讲述了大量数据库实例,但是这些实例基本上是针对个别数据库组件建立的。本章通过几个数据库应用程序的综合实例,进一步说明开发数据库应用程序的方法。

### 8.1 建立主从表数据库应用程序

Delphi 6 提供了一个向导工具,可以非常方便地建立主从表数据库应用程序。该向导工具主要用于建立使用 BDE 的数据库应用程序,如果要使用 ADO 等数据库组件建立应用程序,则可以将 BDE 建立的应用程序中的数据库组件替换为 ADO 等数据库组件即可。下面就通过该向导工具建立一个主从表数据库应用程序。然后再在该应用程序的基础上建立第二层主从表的关联关系,此时第一个从属表会变为第二个从属表的主表。这三个基表是 Delphi 6 提供的 DEMEMOS 数据库中的 Customer.db、Orders.db 和 Items.db 等三个数据库,其中, Customer.db 是 Orders.db 的主表,我们通过向导工具建立该主从表的结构;而 Orders.db 又是 Items.db 的主表,此时需要手工建立该主从表的结构。

#### 8.1.1 使用向导工具建立主从表的关联

我们通过向导工具实现第一层 Customer.db 和 Orders.db 之间的关联。其实现步骤如下:

1. 要使用 Form Wizard 向导工具建立主从表,则必须先建立一个应用程序。然后再选择 Database/Form Wizard 菜单,会打开建立主从表应用程序向导工具的第一屏,该面板有两组选项,这些选项的意义如下:

(1) Form Options 该组选项用于决定建立应用程序的形式,有两个选项:

- Create a single form 用于建立只使用一个基表的应用程序。
- Create a master/detail form 用于建立主从表应用程序。

(2) Dataset Options 该组选项用于决定使用的数据集类型,也包括两个选项:

- Create a form using TTable objects 使用 BDE 组件面板的 TTable 组件作为数据集。
- Create a Form using TQuery objects 使用 BDE 组件面板的 Query 组件作为数据集。

在向导的第一屏的两组选项中,都选择第二项,此时该面板的显示如图 8-1 所示。

2. 单击 Next 按钮打开第二屏,然后在 Directories 目录列表中打开 D:\Borland\Common Files\Borland Shared\Data 目录,此时左边的列表框中会显示出该目录下所有的基表,从中选择主表文件 Customer.db。此时第二屏的显示如图 8-2 所示。

3. 单击 Next 按钮打开第三屏,此时左边的列表框 Available Fields 中会列出主表 Customer.db 中所有的字段,通过使用 Shift 键,从列表中选择前 7 个字段,然后单击按钮“>”,则这些字段会移动到右边的列表框中,列表框中字段的排列顺序也就是这些字段在表单中显示的先后顺序,可以通过列表框下面的两个上下箭头按钮移动这些字段的排列顺序。此时该屏的显示如图 8-3 所示。

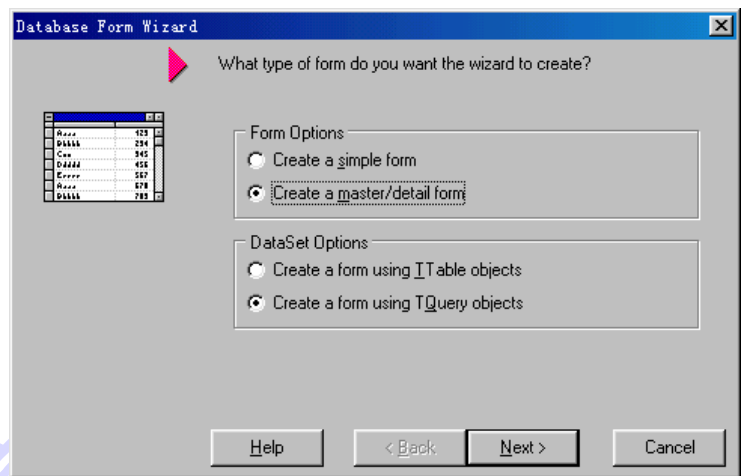


图 8-1 向导第一屏的设置

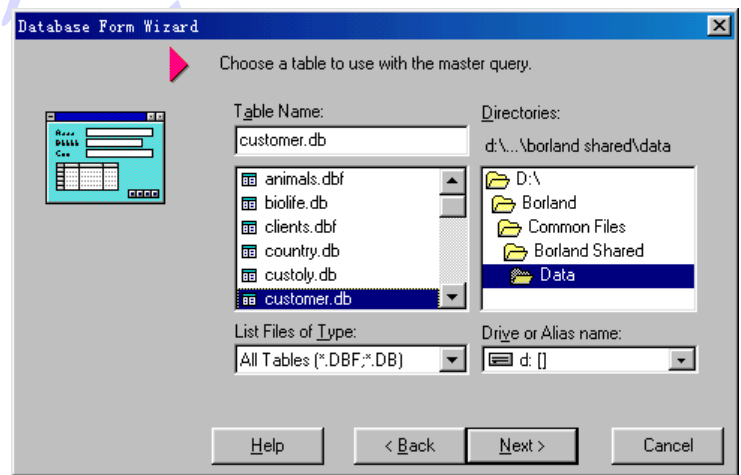


图 8-2 选择主表

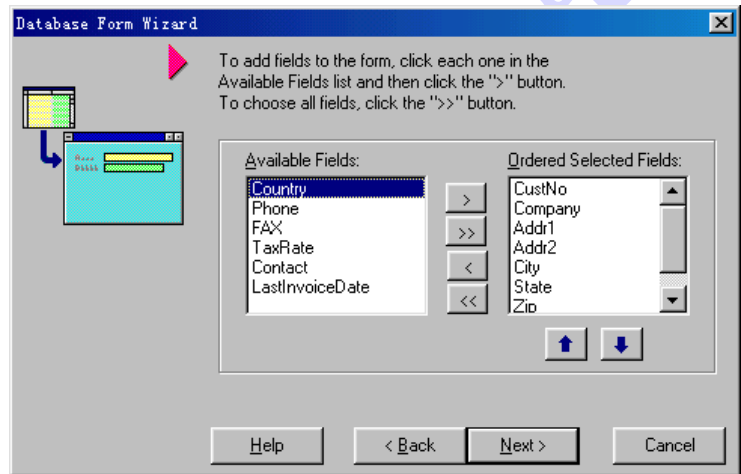


图 8-3 选择显示在表单中的主表字段

4. 单击 Next 按钮可以打开下一屏，该屏有三个单选项，各个选项的意义如下：



- **Horizontally** 主表的字段逐行排列，如果第一行排列不下，则放置到第二行。该选项是默认选项。
  - **Vertically** 主表的字段上下垂直排成一行。每行只有一个字段。
  - **In a grid** 主表的字段按照数据表格的形式显示，即使用 TDBGrid 组件显示。
- 现在使用默认选项，此时该屏的显示如图 8-4 所示。

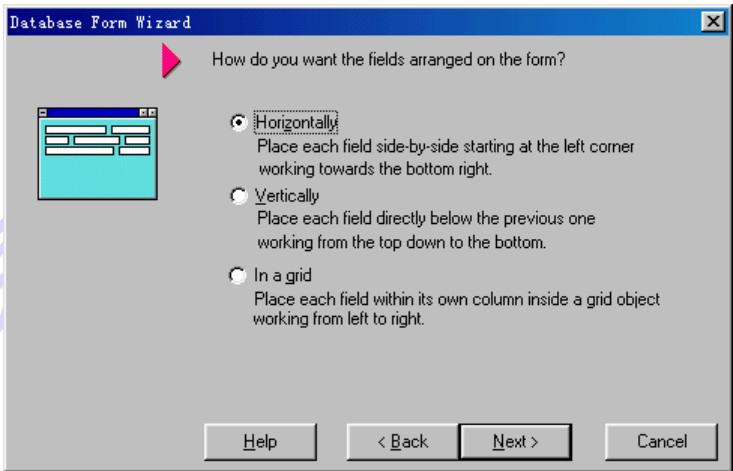


图 8-4 选择主表字段的显示形式

5. 单击 Next 按钮可以打开下一屏，该屏用于选择从属表，从左边的基表列表框中选择 Orders.db，此时该屏的显示如图 8-5 所示。

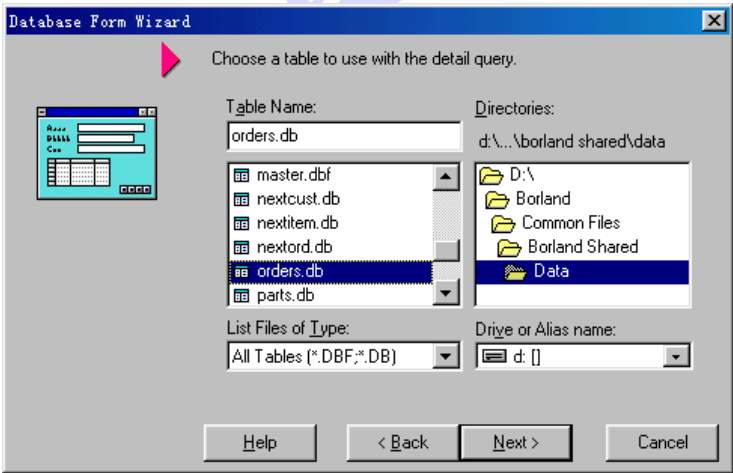


图 8-5 选择从属表

6. 单击 Next 按钮可以打开下一屏，该屏用于选择表单中显示的从属表字段，可以从左边的列表框中选择所有字段，然后单击 “>>” 按钮，则从属表中所有字段都移动到了右边的列表中。此时该屏的显示如图 8-6 所示。

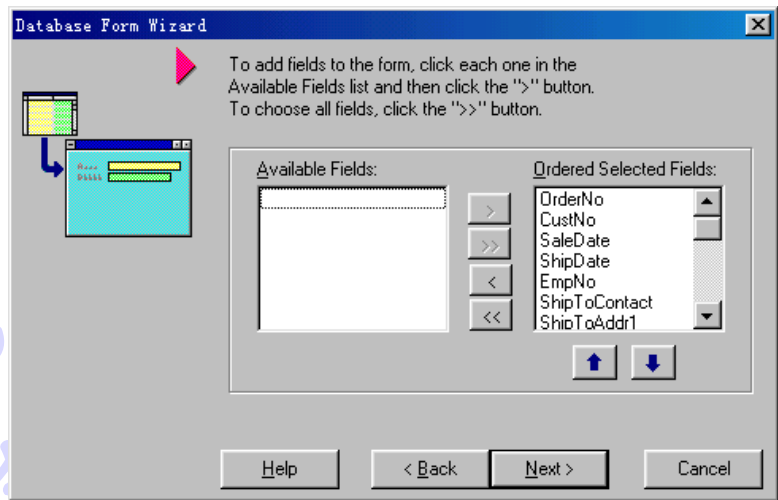


图 8-6 选择从属表的所有字段进行显示

7. 单击 Next 按钮可以打开下一屏，该屏用于选择从属表字段在表单中的显示方式，由于主表的一条记录对应从属表的多条记录，所以应选择默认的选项 In a grid，这样从属表会以数据表格的形式显示所有与主表当前记录关联的记录。如图 8-7 所示。

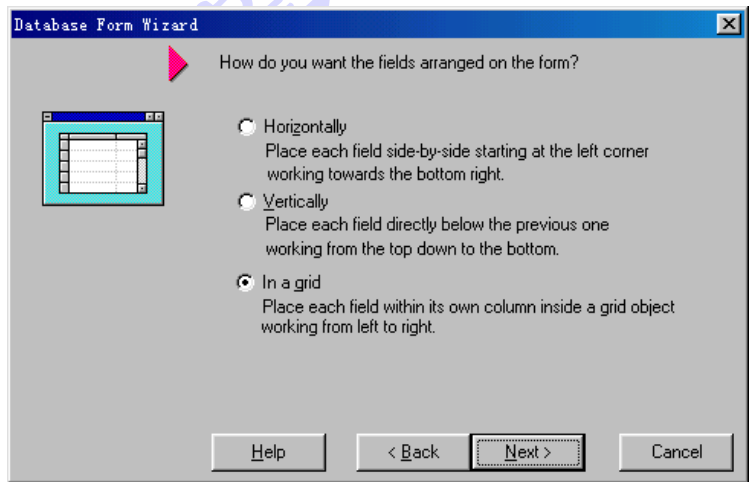


图 8-7 选择从属表记录的显示方式

8. 主从表的记录要关联，则必须有一个以上的关联字段。所以下一屏主要用于设置主从表的关联字段。Customer.db 和 Orders.db 的关联字段是 CustNo，所以应从左边（Detail Fields）从属表的字段中选择该字段，再从右边（Master Fields）主表的所有字段中选择该字段，此时面板中间的 Add 按钮变为可用，单击该按钮，则主从表的关联字段会添加到下面的关联字段列表框（Joined Fields）中，此时该面板的显示如图 8-8 所示。

如果要删除或清除所有的关联字段，则可以单击 Delete 和 Clear 按钮。

9. 单击 Next 按钮，则会打开下一屏，该屏有两部分选项：

- Generate a main form 选择该选项，则在建立应用程序时需要指定一个表单作为主表单。一般不需要选择该选项。

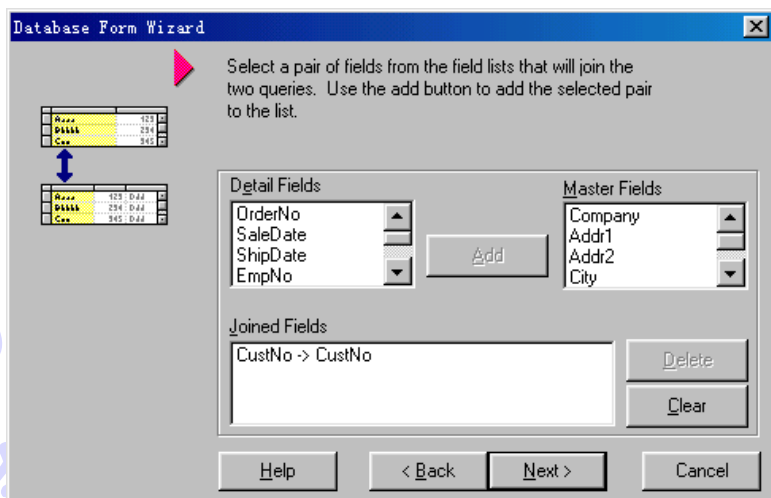


图 8-8 设置主从表的关联字段

- **Form Generation** 设置应用程序建立表单的形式，如果选择 **Form Only** 选项，则只会建立一个表单，所有不可见的数据库组件将放置在该表单上；如果选择 **Form and DataModule** 选项，则应用程序会建立一个表单和一个数据模块，所有不可见的数据库组件将放置在该数据模块上。

该实例中选择 **Form and DataModule** 选项，这样应用程序的层次就比较清晰。

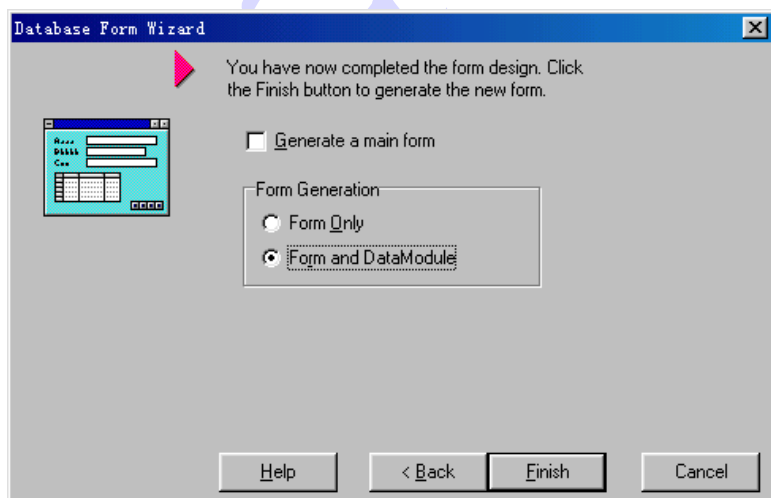


图 8-9 设置应用程序的设计结构

10. 单击 **Finish** 按钮，结束主从表向导，并建立最终的应用程序。最终建立的应用程序主界面如图 8-10 所示，建立的数据模块进行适当调整后，如图 8-11 所示。

11. 由于 TQuery 组件默认情况下建立的应用程序是不能直接对数据进行修改，这是因为其默认的 **RequestLive** 属性为 **False**，该属性决定是否可以对 TQuery 组件的数据进行修改。所以在该应用程序中，将 **Query1** 和 **Query2** 的 **RequestLive** 属性为都设置为 **True**，这样就可以修改主从表的数据了。

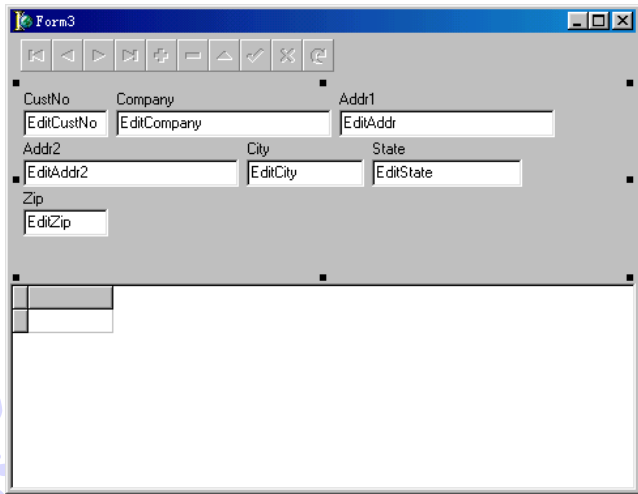


图 8-10 建立的应用程序主界面

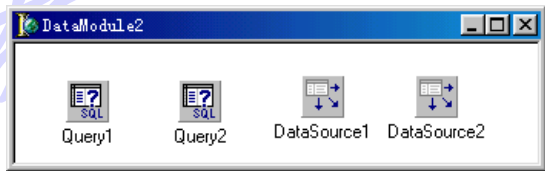


图 8-11 建立的数据模块

12. 由于新建一个应用程序时，默认将 Form1 设置为应用程序的主界面，但是使用主从表向导工具建立的主界面是 Form3，而 Form1 表单中没有放置任何组件，所以该应用程序需要将 Form3 设置为主表单，同时可以删除 Form1。设置方法如下：

选择菜单 Project|Options 打开项目选项面板的 Forms 标签页，在 Main form 选项的下拉框中选择 Form3，此时选项面板的显示如图 8-12 所示，这样就将 Form3 设置为应用程序的主表单，也就是运行应用程序时第一个显示的界面。

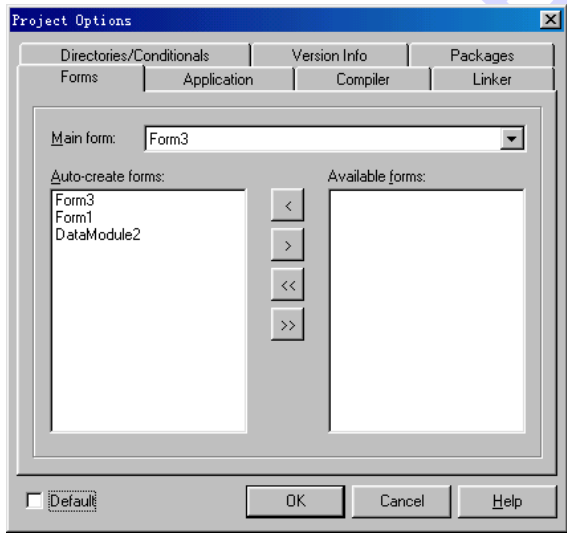


图 8-12 设置应用程序的主表单为 Form3

下面再删除 Form1。选择 Project|Remove from Project...菜单，然后选择 Form1 所在的一行，如图 8-13 所示，单击 OK 按钮后，会弹出一个对话框，再确认后就可以从应用程序中删除该表单及其对应的单元文件 Unit1。

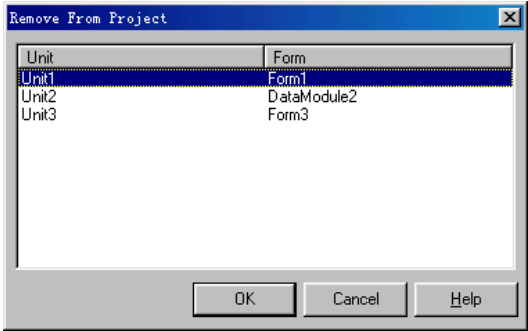


图 8-13 删除表单 Form1

8.1.2 手工建立第二级主从关联

由于前面实例中的从属表 Orders.db 是 Items.db 的主表，所以这两个基表之间又可以建立主从表的关联。手工建立这两个基表间关联的步骤如下：

1. 打开数据模块面板 DataModule2，然后在该面板中添加一个数据集组件 Table1 和一个数据源组件 DataSource3。

再打开表单 Form3，将 ScrollBox 和 Panel3 的 Align 属性都设置为 alNone，这样就可以任意改变这两个组件的大小及位置。另外将 ScrollBox 的 VertScrollBar 属性的 Visible 子属性设置为 False，因为现在不需要滚动条。同时，适当调整各个组件的位置，并将 Form3 的高度增大，以便下面留出空间放置显示 Items 的数据。

在 Form3 的下面添加一个数据表格组件 DBGrid2，新添加的几个数据库组件的属性设置值见表 8-1。

表 8-1 设置新添加的数据库组件的属性

组件	属性	设置值	说明
DataModule2.Table1	Active	True	激活数据集
	DatabaseName	DBDEMOS	
	IndexFieldName	OrderNo	
	MasterSource	DataSource2	设置其主表 Orders.db 使用的数据源
	MasterFields	OrderNo	设置与其主表 Orders.db 关联的字段
	TableName	Items.db	设置使用的基表
DataModule2.DataSource3	DataSet	Table1	
Form3.DBGrid2	DataSource	DataSource2	

注意：该应用程序在建立时，已经在 Form3 对应的 Unit3 单元文件中插入了数据模块 DataModule2 对应单元文件 Unit2 的引用，所以 Form3 中的组件可以直接使用 DataModule2 中定义的组件。该语句如下：

```
uses Unit2;
```

以上组件的属性设置完毕后,表单 Form3 的显示如图 8-14 所示,数据模块 DataModule2 的显示如图 8-15 所示。

Figure 8-14 shows a Windows form titled 'Form3'. It contains several text boxes for customer information: 'CustNo' (with 'EditCustNo' below it), 'Company' (with 'EditCompany' below it), 'Addr1' (with 'EditAddr' below it), 'Addr2' (with 'EditAddr2' below it), 'City' (with 'EditCity' below it), 'State' (with 'EditState' below it), and 'Zip' (with 'EditZip' below it). Below these fields is a large empty rectangular area. At the bottom of the form is a table with the following data:

OrderNo	ItemNo	PartNo	Qty	Discount
1003	1	1313	5	0
1004	1	1313	10	50
1004	2	12310	10	0
1004	3	3316	8	0
1004	4	5324	5	0
1005	1	1320	1	0

图 8-14 调整并添加组件后 Form3 的显示

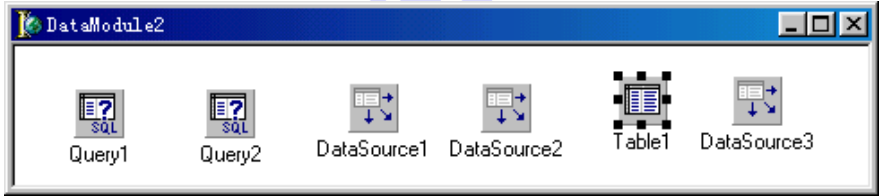


图 8-15 添加新组件后 DataModule2 的显示

2. 现在运行应用程序,则显示出最终的界面,此时使用导航条导航主表的数据时,两级从属表对应的表格中的数据都会相应滚动;如果主表的数据不动,只有导航中间表格 Orders.db 对应的数据,则可以看到 Items.db 对应的表格中的数据也会做相应地改变,只会查询出 OrderNo 与 Orders.db 中相等的记录。

比如,如果主表 Customer.db 中的 CustNo 为 1351,则从属表 Orders.db 对应的数据表格中只会查询出 CustNo 为 1351 的记录;在这些记录中选择 OrderNo 为 1055 的记录,则 Items.db 对应的数据表格中也只会显示出 OrderNo 为 1055 的所有记录,这样就构成了两级主从表关联的关系,此时界面的显示如图 8-16 所示。

3. 从上面的实例可以看出,建立主从表的关键是:

(1) 如果从属表使用的是一个 TQuery 组件,则其 DataSource 属性要设置为主表使用的数据源的名字,同时在 SQL 属性的 Where 子句中要通过关联字段与主表建立关联:比如 Query2 中的 Where 子句为:



Where "D:\Borland\Common Files\Borland Shared\Data\orders.db"."CustNo"  
=: "CustNo"

OrderNo	CustNo	SaleDate	ShipDate	EmpNo	S
1003	1351	1988-04-12	1988-05-03 12:00:00	114	
1052	1351	1989-01-06	1989-01-07	144	
1055	1351	1989-02-04	1989-02-05	29	

OrderNo	ItemNo	PartNo	Qty	Discount
1055	1	2367	8	0
1055	2	2954	7	0
1055	3	12386	7	0
1055	4	13545	5	0

图 8-16 两级主从表的查询结果

其中:"CustNo"是主表的关联字段，注意不要删除前面的冒号“:”，该冒号用来表示这是一个参数。

(2) 如果从属表是 TTable 组件，则应设置其 MasterDataSource 属性值为其主表使用的数据源，将 MasterFields 属性设置为与主表关联的字段名。

掌握了以上关键内容，就可以手工建立主从表关联的应用程序。

## 8.2 建立日程安排应用程序

在实际工作中，我们经常需要制定一些计划、规划等，这些内容使用数据库管理也十分方便，同时这类应用程序大多需要处理日期，所以在该应用程序中，我们重点剖析日期类型数据的处理方法。

该应用程序相对比较复杂一些，它有以下单元文件及表单组成：

- Lead\_Project 整个项目对应的单元文件，用于创建各个表单。
- Head\_Module 用于放置不可见数据库组件的表单 DataModule\_Head 对应的单元文件。
- Head\_Public\_Unit 一个共用的独立单元文件，用于定义一些全局变量及函数，没有对应表单。
- Head\_Main 应用程序主表单 Form\_Head 对应的单元文件。
- Head\_Perday 一天的工作安排表单是 Form\_Perday 对应的单元文件。
- Head\_GbLogin 用户注册窗口 FrmLogin 对应的单元文件。
- Head\_Canend\_Input 日期输入表单 F\_CalendInput 对应的单元文件，可用于计划的

查询，并可以通过该日历打开一天的工作安排。

8.2.1 建立基表

要建立一个应用程序，需要先建立一个数据库。为了方便，我们使用 Interbase 提供的数据库 D:\InterBase\examples\database\employee.gdb，该数据库的别名为 INTRBASE1。该应用程序需要两个基表，一个用于保存计划数据，名字为 leader\_plan；另外基表用于保存用户名和口令，名字为 pwd。使用 Database Desktop 建立基表 leader\_plan 的界面如图 8-17 所示。

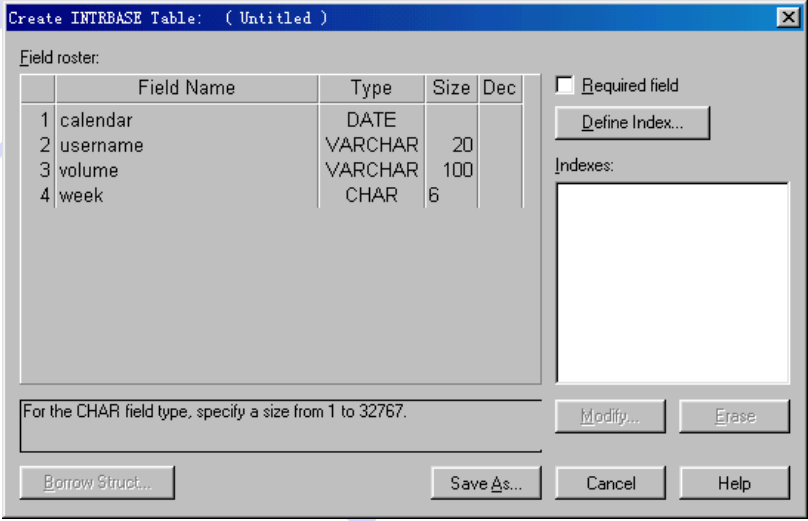


图 8-17 建立基表 leader\_plan

基表建立完成后，选择 File|Save 保存基表结构，并从打开的文件选择器中选择数据库别名 INTRBASE1，然后会弹出一个数据库注册对话框，如图 8-18 所示。在该对话框 password 编辑框中输入 masterkey 口令，则可以登录到该数据库，最后为该基表输入名字 leader\_plan，如图 8-19 所示，则该基表会保存到 employee.gdb 数据库中。

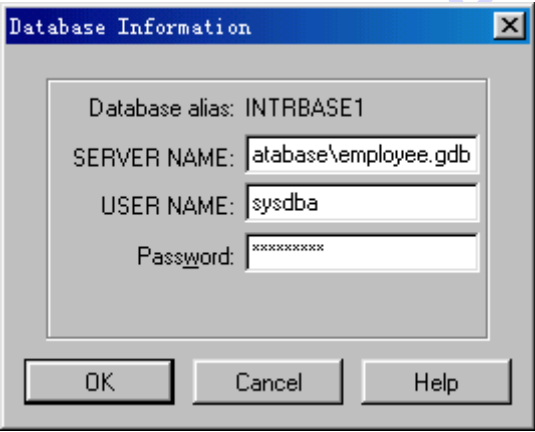


图 8-18 数据库注册窗口

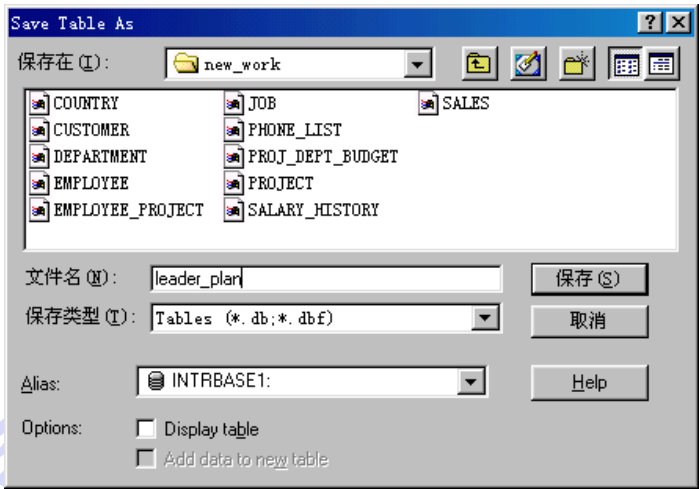


图 8-19 将基表 leader\_plan 保存到 employee.gdb 数据库中

另一个基表 user\_pwd 与 leader\_plan 基表的建立及保存方法相似。这两个基表中各个字段的名称、类型、长度及意义，见表 8-2。

表 8-2 基表 leader\_plan 和 user\_pwd 的定义

基表名称	字段名称	字段类型	字段长度	说明
Leader_plan	calendar	date		计划的日期
	username	varchar	20	用户保存用户名， 因为一个基表可以 保存多个用户的计 划，所以每个用户 都需要有一个用户 名，该用户名与 user_pwd 基表中的 name 相对应
	volume	varchar	100	计划的内容
	week	char	6	计划对应一周的哪 一天
User_pwd	id	short		设置用户的序列号
	name	varchar	20	保存用户名
	pwd	varchar	20	保存用户的口令

8.2.2 建立应用程序数据模块

建立应用程序主界面的步骤如下：

1. 使用 File|New|Application 菜单新建立一个应用程序。
2. 使用 File|New|Data Module 菜单，在该应用程序中新添加一个数据模块，将该数据模块的名字修改为 DataModule\_head。
3. 向数据模块中添加组件。通过 BDE 组件面板，在该数据模块中添加一个数据库组

件 Database1，添加两个基表组件 Table\_head 和 usr\_pwd，添加两个数据源组件 DataSource\_head 和 DataSource\_pwd，此时该数据模块的显示如图 8-20 所示。

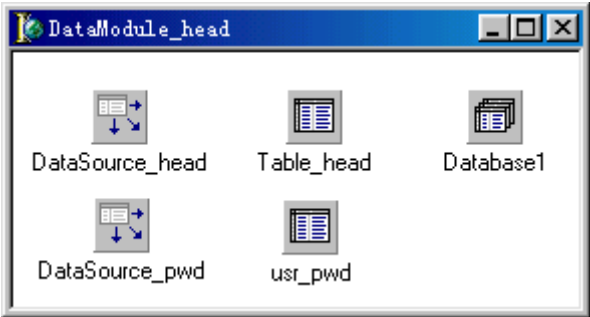


图 8-20 添加组件后的数据模块

4. 设置数据模块中组件的属性。各个组件的属性设置值见表 8-3。

表 8-3 设置数据模块中组件的属性

组件	属性	设置值	说明
Database1	AliasName	INTRBASE1	设置数据库别名
	DatabaseName	Database1	设置数据库的名字
	LoginPrompt	False	关闭默认的用户注册窗口，而使用新建立的注册窗口 frmlogin
	Params	USER NAME=sysdba PASSWORD=masterkey	设置默认的用户名及口令，可以使用向下的箭头来添加一条新参数
table head	DatabaseName	Database1	
	TableName	Leader_plan	
usr_pwd	DatabaseName	Database1	
	TableNam	user_pwd	
DataSource_head	DataSet	table_head	
DataSource_pwd	DataSet	usr_pwd	

5. 添加处理事件的代码。数据模块对应的单元文件名字为 head\_module，其代码如下：

```
..... //省略单元文件头
var
    DataModule_head: TDataModule_head;

g_username,g_password:string; //两个全局变量，用于保存用户名和口令
sign_1:integer; //全局变量，作为标志符

const //在db库文件中定义的两个错误代码，应在uses中添加db库文件
    eKeyViol=9729; //用于记录的唯一性检查
    eReqdErr=9732; //用于不能为空的字段
implementation
```

```

{$R *.DFM}

procedure TDataModule_head.Table_headBeforePost(DataSet: TDataSet);
begin //提交数据前触发该事件, 检查calendar和username两个字段对应的编辑框不为空
    if (Table_headCALENDAR.Text='') or
        (Table_headUSERNAME.Value='') then
        Enull_error.Create('日期不能为空值!');
end;

procedure TDataModule_head.Table_headPostError(DataSet: TDataSet;
    E: EDatabaseError; var Action: TDataAction);
begin //提交数据如果遇到唯一性冲突及非空字段为空的错误, 则出现异常
    if (E is EDBEngineError) then
        if (E as EDBEngineError).Errors[0].Errorcode = eKeyViol then
            begin
                MessageDlg('不能提交: 不能有两个相同的日期!', mtWarning, [mbOK], 0);
                abort;
            end
        else if (E as EDBEngineError).Errors[0].Errorcode = eReqdErr then
            begin
                MessageDlg('不能提交: ''日期''值不能为空!', mtWarning, [mbOK], 0);
                abort;
            end;
end;

//下面的事件在使用导航条删除记录时触发, 会弹出一个中文提示框, 此时需
//要将主表单导航条 (DBNavigator1) 的ConfirmDelete属性设置为False
procedure TDataModule_head.Table_headBeforeDelete(DataSet: TDataSet);
begin
    if MessageDlg('确实要删除吗?', mtConfirmation,
        [mbYes, mbNo], 0) = mrno then
        abort;
end;

//该事件是对应CALENDAR字段的事件, 用于检查输入数据的有效性
procedure TDataModule_head.Table_headCALENDARSetText(Sender: TField;
    const Text: String);
begin
    try
        strtodate(Text); //将输入的日期转换为字符串
    except
        raise exception.create(''+text+'' 是一个无效的日期值!'); //提交一个异常
    abort;
    end;
    Sender.Asstring:=text;
end;

```

以上4个事件中, 前3个都是 Table\_head 基表的事件, 而最后一个则是对应 CALENDAR 字段的事件。设置该事件的方法如下:

在 DataModule\_head 中选择 Table\_head 基表, 然后上面单击鼠标右键, 从弹出的上

下文菜单中选择 Fields Editor 菜单，则会打开基表的字段编辑器。在该编辑器中再单击鼠标右键，从弹出菜单中选择 Add All Fields，则基表 leader\_plan 中所有的字段都添加到了该字段编辑器中，如图 8-21 右边所示。在该字段编辑器中选择 CALENDAR 字段，然后打开对象观察器的 Events 标签页，在 OnSetText 事件上面双击鼠标，则会添加一个新的事件，再添加前面说明的代码即可。

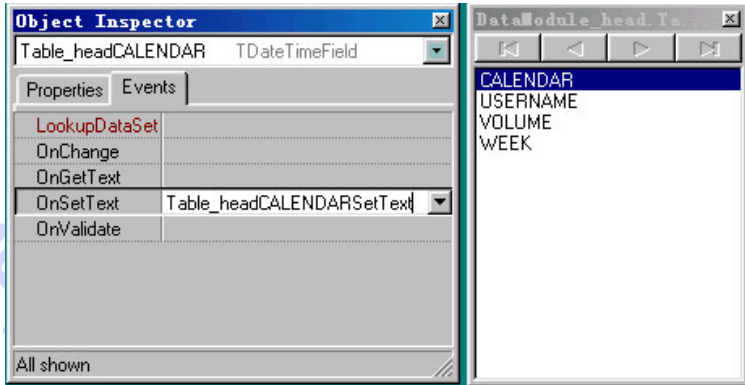


图 8-21 字段编辑器及 CALENDAR 字段的事件面板

### 8.2.3 设计主表单 form\_head

#### 1. 设计主表单的界面。

为了使该表单可以使用 DataModule\_head 中定义的数据库组件，应在 form\_head 中使用 File|Use Unit 菜单来引用 DataModule\_head 的表单 head\_module。

(1) 添加组件。在 Form1 的名称修改为 form\_head，然后添加如下组件：

在页面的顶部添加两个面板 Panel1 和 Panel2，然后在 Panel1 上添加页面标题“日常工作安排”，在 Panel2 上添加标签 Label2、Label6、Label7、Label8、Label9，分别将其 Caption 属性设置为“2001”、“.”、“07”、“.”、“09”，这些标签用来表示当前的日期。

在表单中添加一个面板 Panel3，将其拖拉到比较大，以便用于放置 DBCtrlGrid1 等，该组件主要作用是使页面更美观。

打开 Data Controls 组件面板，选择 TDBCtrlGrid 组件，然后在 Panel3 中添加一个数据控制表格组件 DBCtrlGrid1。

在 DBCtrlGrid1 的第一行添加一个 TDBText 组件 DBText\_week，用于显示星期，该字段不需要输入，在输入完日期字段后自动计算；添加一个 TDBEdit 组件 DBEdit\_name，用于输入用户名，该编辑框被隐藏起来，在输入完其他数据后按照当前注册的用户来插入该用户的用户名；再添加一个 TDBEdit 组件 DBEdit\_date，用于输入或显示该工作安排的日期；最后添加一个 TDBMemo 组件 DBMemo\_topic，用于显示工作安排内容。然后将 DBEdit\_date 和 DBMemo\_topic 编辑框的字体设置为“仿宋|粗体|小四”。

在表单的左上角添加一个 TDBEdit 组件 DBEdit\_user，用来保存用户在口令注册窗口输入的用户名。

另外，在 DBCtrlGrid1 的上面添加三表标签，分别将其 Caption 属性设置为“星期”、



“日期”、“工作内容”，并设置合适的字体，调整到对应基表三个字段的位置。

在面板 Panel3 的底部添加一个 TDBNavigator 组件 DBNavigator1，用于数据库数据的导航。

最后，在面板 Panel3 的右下角添加 5 个 TBitBtn 按钮，分别设置合适的图标，这些图标位于 ...\\Common Files\\Borland Shared\\Buttons。将这些按钮的名字分别修改为 btn\_calendar、btn\_thisweek、btn\_lastweek、btn\_nextweek、Btn\_exit，将其 Caption 属性分别修改为“日历”、“本周”、“上周”、“下周”、“退出”。

以上所有组件添加完毕并适当调整其位置和大小后，主表单的显示如图 8-22 所示。



图 8-22 主表单的显示

(2) 设置一些主要组件的属性。几个主要组件属性的设置值见表 8-4。

表 8-4 主表单几个主要组件属性的设置

组件	属性	设置值	说明
DBEdit_user	DataSource	DataModule_head.DataSource_pwd	
	DataField	Name	
DBCtrlGrid1	DataSource	DataModule_head.DataSource_head	
	RowCount	5	该组件重复显示的行数
DBEdit_name	DataSource	DataModule_head.DataSource_head	
	DataField	USERNAME	
	Visible	False	在运行时不显示该字段
DBText_week	DataSource	DataModule_head.DataSource_head	
	DataField	WEEK	
DBEdit_date	DataSource	DataModule_head.DataSource_head	
	DataField	CALENDAR	

(续表)

组件	属性	设置值	说明
DBMemo_topic	DataSource	DataModule_head.DataSource_head	
	DataField	VOLUME	
DBNavigator1	DataSource	DataModule_head.DataSource_head	
	ConfirmDelete	False	将该属性设置为 False, 在使用删除按钮删除记录时, 不会再显示默认的提示框

2. 添加代码。主表单对应的单元文件是 head\_main, 其代码如下:

```
unit head_main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, Db, DBTables, StdCtrls, ExtCtrls, Buttons, DBCtrls, Mask,
  DBCGrids;
type
  TForm_head = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    DBEdit_user: TDBEdit;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    DBCtrlGrid1: TDBCtrlGrid;
    DBEdit_date: TDBEdit;
    DBMemo_topic: TDBMemo;
    DBEdit_name: TDBEdit;
    DBNavigator1: TDBNavigator;
    btn_calendar: TBitBtn;
    btn_thisweek: TBitBtn;
    btn_lastweek: TBitBtn;
    btn_nextweek: TBitBtn;
    Btn_exit: TBitBtn;
    Panel2: TPanel;
    Label2: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    DBText_week: TDBText;
    panel3: TPanel;

    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure btn_calendarClick(Sender: TObject);
    procedure btn_thisweekClick(Sender: TObject);
    procedure btn_lastweekClick(Sender: TObject);
  end;
end;
```

```

    procedure btn_nextweekClick(Sender: TObject);
    procedure btn_exitClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure DBEdit_dateEnter(Sender: TObject);
    procedure DBEdit_dateDblClick(Sender: TObject);
    procedure DBEdit_dateExit(Sender: TObject);

private
    { Private declarations }
public
    { Public declarations }
    procedure MyException(Sender:TObject;E:Exception);
end;

var
    form_head:Tform_head;

implementation

uses head_gblogin, head_calend_input,head_module,head_public_unit;//调用
其他单元文件

{$R *.DFM}

procedure Tform_head.MyException(Sender:TObject;E:Exception);//通用异常处
理函数
begin
    if E is EVariantError then
        showmessage('无效日期值!')
    else if E is EDBEditError then
        showmessage('无效日期值!')
    else if E is EDatabaseError then
        showmessage('数据库错误!')
    else Application.ShowException(E);
end;

procedure Tform_head.FormCreate(Sender: TObject);
begin
    Application.OnException:=MyException;    //在建立表单时创建该异常函数
end;

procedure Tform_head.FormShow(Sender: TObject);    //显示主表单
var
    a_year,a_month,a_day:word;
begin
    DataModule_head.usr_pwd.Filter:='name='+'''+g_username+''';    //用于
    获得用户名
    DataModule_head.usr_pwd.open;

    //下面4个语句从系统当前日期中提取年、月、日的数值

```

```

    DecodeDate(date,a_year,a_month,a_day);
    last_year:=a_year;
    last_month:=a_month;
    last_day:=a_day;

    //下面6个语句更新表单右上角的日期
    label2.Caption:=IntToSTR(last_year);
    label7.Caption:=IntToSTR(last_month);
    label9.Caption:=IntToSTR(last_day);
    label2.Refresh;
    label7.Refresh;
    label9.Refresh;

    get_filter(a_year,a_month,a_day); //查询本周数据，该函数在
    head_public_unit中定义
end;

procedure TForm_head.btn_calendarClick(Sender: TObject); //单击“日历”按钮
begin
    glabel_str:='query'; //为全局变量设置一个字符串来表示目前是查询操作
    F_calendinput.showmodal; //显示日历表单
end;

procedure TForm_head.btn_thisweekClick(Sender: TObject); //单击“本周”按钮
var
    a_year,a_month,a_day:word;
begin
    DecodeDate(date,a_year,a_month,a_day);
    last_year:=a_year;
    last_month:=a_month;
    last_day:=a_day;
    label2.Caption:=IntToSTR(last_year); //显示年份的标签
    label7.Caption:=IntToSTR(last_month); //显示月份的标签
    label9.Caption:=IntToSTR(last_day); //显示具体某日的标签
    label2.Refresh; //下面三个语句对标签进行刷新
    label7.Refresh;
    label9.Refresh;

    get_filter(a_year,a_month,a_day); //按照本周的日期查询工作安排内容
end;

procedure TForm_head.btn_lastweekClick(Sender: TObject); //单击“上周”按钮
var
    year_temp,month_temp,day_temp:word;
begin
    if last_day<=7 then //***当前日期是1号到7号以内，执行下面的代码***
    begin

```

```

if last_month=1 then    //如果数据显示的时间是1月份时执行下面的代码
begin //由于当前时间是1月份的1号到7号内的日期，所以上周是去年12月
    year_temp:=last_year-1;
    month_temp:=12;
    day_temp:=DayPerMonth(year_temp,month_temp)+last_day-7;
end
else                    //如果据显示的时间不是1月份执行这段代码
begin
    year_temp:=last_year; //由于当前时间不是1月份，所以上周一一定是本年度
    month_temp:=last_month-1;
    day_temp:=DayPerMonth(year_temp,month_temp)+last_day-7;
end;
end
else                    //当前日期不是1号到7号以内，执行下面的代码
begin //上周一定是本年、本月
    year_temp:=last_year;
    month_temp:=last_month;
    day_temp:=last_day-7;
end ;

get_filter(year_temp,month_temp,day_temp); //按照上周时间执行查询

last_year:=year_temp; //下面三个语句保存最新年、月、日
last_month:=month_temp;
last_day:=day_temp;

label2.Caption:=IntToSTR(last_year); //下面刷新主表单右上角的日期
label7.Caption:=IntToSTR(last_month);
label9.Caption:=IntToSTR(last_day);
label2.Refresh;
label7.Refresh;
label9.Refresh;
end;

procedure TForm_head.btn_nextweekClick(Sender: TObject); //单击“下周”按钮
var
    year_temp,month_temp,day_temp:word;
    minus_day:integer;
begin
    minus_day:=(last_day+7)-DayPerMonth(last_year,last_month);
    if minus_day>0 then
    begin
        if last_month=12 then
        begin
            year_temp:=last_year+1;
            month_temp:=1;
            day_temp:=minus_day;
        end;
        else

```

```

        begin
            year_temp:=last_year;
            month_temp:=last_month+1;
            day_temp:=minus_day;
        end;
    end
else
    begin
        year_temp:=last_year;
        month_temp:=last_month;
        day_temp:=last_day+7;
    end;
    get_filter(year_temp,month_temp,day_temp);
    last_year:=year_temp;
    last_month:=month_temp;
    last_day:=day_temp;

    label2.Caption:=IntToStr(last_year);
    label7.Caption:=IntToStr(last_month);
    label9.Caption:=IntToStr(last_day);
    label2.Refresh;
    label7.Refresh;
    label9.Refresh;
end;

procedure TForm_head.Btn_exitClick(Sender: TObject); // “退出”按钮
begin
    Application.Terminate;
end;

procedure TForm_head.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    DataModule_head.Table_head.Close;
    Application.Terminate;
end;

procedure TForm_head.DBEdit_dateDblClick(Sender: TObject); //双击日期编辑框
begin
    sign_1:=2;
    if DBEdit_date.Text<>'' then //如果已经输入日期，则使用该日期打开日历
    begin

        DeCodeDate(DBEdit_date.Field.Value,last_year,last_month,last_day);
        end;
        goble_str:='head_sevenday'; //设置当前是在一周的界面中进行日期的输入
        F_calendinput.Showmodal; //打开日历来输入日期
    end;

    procedure TForm_head.DBEdit_dateExit(Sender: TObject); //退出该字段则显示是
    星期几

```



```

const
    day_no: array[1..7] of
string=(' (日)', ' (一)', ' (二)', ' (三)', ' (四)', ' (五)', ' (六) ');
var
    week_no: integer;
begin
    if (DBEdit_date.Text<>'') and (sign_1<>2) then
    begin
        DBEdit_name.text:=form_head.DBEdit_user.text; //输入当前用户名
        week_no:=DayOfWeek(DBEdit_date.Field.Value);
        DBText_week.Field.Text:=day_no[week_no]; //输入当前日期对应的星期数
        DBEdit_date.Refresh;
    end;
end;

procedure TForm_head.DBEdit_dateEnter(Sender: TObject);
begin
    DataModule_head.Table_head.Edit; //使日期字段可以编辑
end;

end.

```

#### 8.2.4 设计注册窗口

在应用程序注册时，我们不希望使用 Delphi 提供的默认注册窗口，而希望使用界面友好的注册窗口。

1. 添加组件并设置属性。该窗口的设计步骤如下：

通过菜单 File|New|Form，在项目中新添加一个表单，拖拉该表单的边框将其拖拉到较小一些。然后为该表单添加一个标题“用户登录”，并设置为红色。

再添加一个面板，将面板的 BevelOuter 属性设置为 bvRaised，以便使面板有凸出感觉。在面板中加入两个标签，两个编辑框，将标签的 Caption 属性修改为“姓名：”、“口令：”，将编辑框的名字修改为 edit\_username、edit\_password，删除两个编辑框 Text 属性中的内容，再将 edit\_password 的 PasswordChar 属性设置为一个星号“\*”，这样用户输入的口令将以一串星号显示，以便保证口令的安全性。

在面板的下面添加一个标签，将其名字修改为 lblconnect，将其 Caption 中的内容删除。该标签的作用是在连接数据库时提示用户等待。



图 8-23 用户注册窗口

最后，在面板的底部添加两个 TBitBtn 按钮，将其 Kind 属性分别设置为 bkOK、bkCancel，Caption 属性分别设置为“确定”、“取消”，Name 属性分别修改为 btn\_ok、btn\_cancel。

该界面设计完后的显示如图 8-23 所示。

2. 添加代码。该表单对应的核心代码如下：

```
.....//省略文件头
implementation

uses head_main, head_module;           //调用其他单元文件

{$R *.DFM}

procedure Tfrmlogin.btn_cancelClick(Sender: TObject);
begin
    Application.Terminate;
end;

procedure Tfrmlogin.btn_okClick(Sender: TObject);
begin
    frmlogin.lblconnect.caption:='正在连接数据库,请稍候...';    //提示用户
    frmlogin.refresh;
    frmlogin.btn_ok.Cursor:=crSqlWait;           //光标变为沙漏的形状
    DataModule_head.Databasel.Params.Clear;
    //下面两条语句向数据库中添加参数
    DataModule_head.databasel.params.Add('USER
NAME='+Frmlogin.edit_username.Text);

    DataModule_head.databasel.params.Add('PASSWORD='+Frmlogin.edit_password.
Text);

    g_username:=LowerCase(Frmlogin.edit_username.Text); //用户名保存到一个全局
变量中

    try          //正确连接时执行的代码
        DataModule_head.databasel.connected:=TRUE;
        if DataModule_head.databasel.connected then
            begin
                frmlogin.lblconnect.caption:='';
                frmlogin.close;
            end;
        except          //处理未能连接上数据库时的异常
            on EDBEngineError do
                begin
                    showmessage('未能连上数据库!');
                    frmlogin.lblconnect.caption:='';
                    frmlogin.refresh;
                    frmlogin.btn_ok.Cursor:=crDefault;
                end;
            end;
        end;
end;
```

```
end;  
  
end.
```

8.2.5 建立日历窗口

下面建立一个类似于 Windows 控制面板中的一个日历，该日历窗口的实现步骤如下：

1. 添加组件。新建立一个表单，将其名字修改为 F\_calendinput。然后在该表单中添加如下组件：

在面板顶部添加一个面板。

再在该面板上添加一个文本编辑框 Edit1，用于显示年份；在编辑框的右边添加一个上下卷滚条 ScrollBar1，以便改变年份的值。

在面板上的右边添加一个下拉列表框 ComboBox1，用于显示月份。

通过 Samples 面板的 TCalendar 组件在该窗口中添加一个日历组件，将其名字修改为 Calend1，该组件用于显示日期及星期。将该组件的 Year（年）、Month（月）、Day（日）等属性分别修改为 2001、7、31。

最后，在窗口的底部添加两个 TBitBtn 组件，将其名字修改为 btn\_cal\_ok、btn\_cal\_ok，将其 Kind 属性分别修改为 bkOK、bkCancel，将 Caption 属性分别修改为“确定”、“取消”。

以上所有组件添加完毕并设置完属性后，该窗口的显示如图 8-24 所示。



图 8-24 设计完成的日历窗口

2. 添加代码。该表单对应的代码如下：

```
unit head_calend_input;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
  StdCtrls, Mask, DBCtrls, Grids, ExtCtrls, Buttons, ComCtrls, Calendar;  
  
type
```

```

TF_calendinput = class(TForm)
  ScrollBar1: TScrollBar;
  ComboBox1: TComboBox;
  Panel1: TPanel;
  Edit1: TEdit;
  btn_cal_ok: TBitBtn;
  btn_cal_cancle: TBitBtn;
  Calend1: TCalendar;

  procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  procedure ComboBox1Change(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormActivate(Sender: TObject);
  procedure btn_cal_okClick(Sender: TObject);
  procedure btn_cal_cancleClick(Sender: TObject);
  procedure Calend1DbClick(Sender: TObject);

private
  { Private declarations }
public
  { Public declarations }
end;

var
  F_calendinput: TF_calendinput;

implementation

uses head_main, head_module, head_perday, head_public_unit;

{$R *.DFM}

//年度滚动条ScrollBar1滚动时执行下面的代码
procedure TF_calendinput.ScrollBar1Scroll(Sender: TObject; ScrollCode:
TScrollCode;
  var ScrollPos: Integer);
begin
  if (Calend1.month=2) and LeapYear(Calend1.year) and (Calend1.day=29)
  then
    Calend1.day:=28;
  if ScrollCode=scLineUp then      //向上滚动
    Calend1.year:=StrToInt(Edit1.text)+1;
  if ScrollCode=scLineDown then    //向下滚动
    Calend1.year:=StrToInt(Edit1.text)-1;
  Edit1.text:=IntToStr(Calend1.year); //输入年度值
  Calend1.UpdateCalendar;      //更新日历
end;

```

```

//月份下拉框中的月份改变时触发下面的事件
procedure TF_calendinput.ComboBox1Change(Sender: TObject);
var
    day_now, month_now, year_now:integer;
begin
    day_now:=Calend1.day;           //通过TCalendar组件获得当前的天数
    Calend1.day:=28;                //将日历中的天数设置为28天
    Calend1.month:=ComboBox1.itemindex+1; //ComboBox1的itemindex属性从0开始

    Month_now:=Calend1.Month;
    Year_now:=Calend1.Year;
    if DayPerMonth(year_now,month_now)<day_now then
        Calend1.Day:=DayPerMonth(year_now,month_now)
    else
        Calend1.day:=day_now;

    Calend1.UpdateCalendar;
end;

//当年份的数值改变时触发下面的事件
procedure TF_calendinput.Edit1Change(Sender: TObject);
var
    day_now:integer;
begin
    try
        //正常执行的代码
        day_now:=Calend1.day;
        Calend1.day:=28;
        Calend1.year:=StrToInt(Edit1.text);
        if (Calend1.month=2) then //2月份
            begin
                if LeapYear(Calend1.year) and (day_now=29) then //该年为闰年
                    Calend1.Day:=29
                else Calend1.day:=28;
            end
        else Calend1.day:=Day_now;

        Calend1.UpdateCalendar;
    except
        //处理异常
        on EConvertError do //年份应在1-9999之间
            MessageDlg('请输入1-9999之间的数字!', mtError, [mbok], 0);
    end;
end;

procedure TF_calendinput.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    F_calendinput.Close;
end;

```

```

        //每次激活日历窗口时触发下面的事件
procedure TF_calendinput.FormActivate(Sender: TObject);
var
    ayear, amonth, aday: word;
begin
    if glocale_str='head_sevenday' then //通过全局变量确定显示的界面是一周的界面
    begin
        Edit1.text:=IntToStr(last_year);
        ComboBox1.ItemIndex:=last_month-1;
        Calend1.year:=last_year;
        Calend1.month:=last_month;
        Calend1.day:=last_day;
        Calend1.UpdateCalendar;
    end
    else //此时单击了主表单的“日历”按钮,即glocale_str='query'
    begin
        decodedate(date, ayear, amonth, aday);
        Calend1.year:=ayear;
        Calend1.month:=amonth;
        Calend1.day:=aday;
        Edit1.text:=IntToStr(ayear);
        ComboBox1.ItemIndex:=amonth-1;
    end;
end;

//单击“确定”按钮时触发下面的事件
procedure TF_calendinput.btn_cal_okClick(Sender: TObject);
const
    day_no: array[1..7] of
string=(' (日)', ' (一)', ' (二)', ' (三)', ' (四)', ' (五)', ' (六) ');
var
    week_no: integer;
    str_temp: string;
begin
    if glocale_str='head_sevenday' then //显示界面是一周的界面,即主界面
    begin
        DataModule_head.Table_head.edit; //将基表设置为可编辑状态,以便输入内容

        str_temp:=DateToStr(EncodeDate(Calend1.year, Calend1.month, Calend1.day));
        form_head.DBEdit_date.Refresh;
        if form_head.DBEdit_date.Text<>'' then
        begin //如果主界面的日期字段不为空,则会按照该字段的值显示“日历”窗口
            form_head.DBEdit_name.text:=form_head.DBEdit_user.text;

            week_no:=DayOfWeek(EncodeDate(Calend1.year, Calend1.month, Calend1.day));
            form_head.DBEdit_date.Field.Text:=str_temp;
            form_head.DBText_week.Field.Text:=day_no[week_no];
        end
    end
end;

```



```

else if glocale_str='head_oneday' then      //显示的是一天工作安排的界面
begin
    DataModule_head.Table_head.edit;
    form_perday.DBE_Pdate.Text:=DateToStr(EncodeDate(Calend1.year
                                                ,Calend1.month,Calend1.day));
    if form_perday.DBE_Pdate.Text<>' ' then
    begin
        form_perday.DBE_Pname.text:=form_head.DBEdit_user.text;

week_no:=DayOfWeek(EncodeDate(Calend1.year,Calend1.month,Calend1.day));
        form_perday.DBE_Pweek.Field.Text:=day_no[week_no];
    end;
end
else if glocale_str='query' then //通过主表单的“日历”按钮打开
F_calendinput
begin
    Form_head.show;
    last_year:=calend1.year;
    last_month:=calend1.month;
    last_day:=calend1.day;
    get_filter(Calend1.year,Calend1.month,Calend1.day);
end;

//下面3个全局变量用于保存最后的年、月、日，它们在head_public_unit中定义
last_year:=Calend1.Year;
last_month:=Calend1.month;
last_day:=Calend1.day;

form_head.label2.Caption:=IntToStr(last_year);
form_head.label7.Caption:=IntToStr(last_month);
form_head.label9.Caption:=IntToStr(last_day);
form_head.label2.Refresh;
form_head.label7.Refresh;
form_head.label9.Refresh;
end;

procedure TF_calendinput.btn_cal_cancelClick(Sender: TObject);
begin
    F_calendinput.Close;
end;

//下面是在显示一个月中天数的组件Calend1上双击鼠标时触发的事件
procedure TF_calendinput.Calend1DbClick(Sender: TObject);
begin
    if glocale_str='query' then //通过主表单form_head的“日历”按钮打开
F_calendinput
        form_perday.Showmodal; //此时会显示一天的工作安排界面
    end;

end.

```

8.2.6 设计一天工作安排窗口

当用户通过主界面的“日历”按钮打开“日历”窗口 F\_calendinput 时，我们希望用户双击该窗口下面的天数列表时，能够打开一天工作安排的窗口，以便在工作内容较多时逐日浏览工作内容。同时使用该界面输入工作内容也更方便。

该窗口的实现步骤如下：

1. 添加组件。

新建一个表单，将其 Name 属性修改为 Form\_perday，Caption 修改为“一天工作安排”。

通过 TDBEdit 组件在页面中添加两个文本编辑框，分别将其名字修改为 DBE\_Pdate、DBE\_Pname；通过 TDBText 组件在页面上添加一个数据文本框，将其名字设为 DBE\_Pweek，该字段的值通过日期中的输入值获得。分别在 DBE\_Pdate 和 DBE\_Pweek 的前面添加两个标题为“日期”和“星期”。

使用 TDBMemo 组件在页面的中央位置添加一个多行文本编辑框，将其名字修改为 DBM\_Pvolume。

在页面的底部添加一个数据导航组件 DBNavigator1。

再在页面的右下角添加 4 个 TBitBtn 按钮，为这些按钮设置合适的图标，再将这些按钮的名字分别命名为“btn\_today”、“btn\_last”、“btn\_next”、“btn\_exit”，最后将其 Caption 属性分别修改为“今天”、“昨天”、“明天”、“退出”。

注意：该页面的“昨天”与“明天”是一个相对与当前显示日期的概念，不是我们通常所说的“今天”的前一天或后一天。比如，连续 3 次单击“昨天”按钮，则会显示页面当前日期前 3 天的内容。

2. 设置组件的属性。

一些主要组件的属性设置值见表 8-5 所示。

表 8-5 一些主要组件的属性设置值

组件	属性	设置值	说明
DBE_Pdate	DataSource	DataModule_head.DataSource_head	设置数据源
	DataField	CALENDAR	设置对应字段
DBE_Pweek	DataSource	DataModule_head.DataSource_head	
	DataField	WEEK	
DBE_Pname	DataSource	DataModule_head.DataSource_head	
	DataField	NAME	
DBE_Pvolume	DataSource	DataModule_head.DataSource_head	
	DataField	VOLUME	
DBNavigator1	DataSource	DataModule_head.DataSource_head	
	VisibleButtons	[nbDelete,nbEdit,nbPost,nbCancel,nbRefresh]	设置可显示的按钮，即将其他按钮的属性设置为 False

以上所有组件的属性设置完毕后，“一天工作安排”设计界面的显示如图 8-25 所示。

## 3. 添加代码。该界面对应的单元文件的完整代码如下：

.....//省略单元文件头

implementation

uses

head\_module, head\_calend, head\_main, head\_calend\_input; //引用其他单元文件

{ \$R \*.DFM }

procedure TForm\_perday.btn\_exitClick(Sender: TObject); // “退出”按钮触发的事件

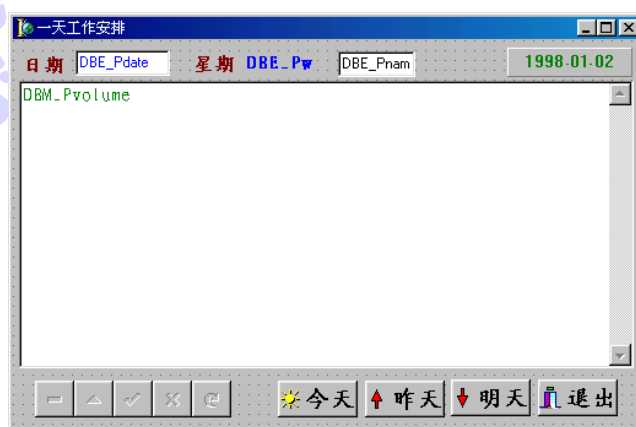


图 8-25 “一天工作安排”的设计界面

begin

form\_perday.Close;

end;

procedure TForm\_perday.FormShow(Sender: TObject); //表单显示时触发的事件

var

ayear, amonth, aday: word;

date\_temp: Tdatetime;

begin

ayear:=F\_calendinput.Calendl.Year;

amonth:=F\_calendinput.Calendl.Month;

aday:=F\_calendinput.Calendl.Day;

date\_temp:=EncodeDate(ayear, amonth, aday);

DataModule\_head.Table\_head.Filter:='username='+'''+form\_head.DBEdit\_use  
r.text

+'''+ ' and calendar='+DateToStr(date\_temp);

DataModule\_head.Table\_head.Open;

last\_year:=ayear;

last\_month:=amonth;

last\_day:=aday;

end;

```

procedure TForm_perday.btn_todayClick(Sender: TObject); //单击“今天”按钮
begin

    DataModule_head.Table_head.Filter:='username='+'''+form_head.DBEdit_use
r.text
                                +'''+ ' and calendar='+DateToStr(Date);
    DataModule_head.Table_head.Open;
    DecodeDate(Date,last_year,last_month,last_day); //将当前日期划分为年、
月、日保存
end;

procedure TForm_perday.btn_lastClick(Sender: TObject); //单击“昨天”按钮
var
    date_temp:TdateTime;
begin
    if last_day=1 then          //当前日期是某月的1号
    begin
        if last_month=1 then //表示当前是1月1日，则“昨天”是去年的12月份
        begin
            last_year:=last_year-1;
            last_month:=12;
        end
        else
            last_month:=last_month-1;

        last_day:=DayPerMonth(last_year,last_month);
    end
    else
        last_day:=last_day-1;

    date_temp:=EncodeDate(last_year,last_month,last_day);

    DataModule_head.Table_head.Filter:='username='+'''+form_head.DBEdit_use
r.text
                                +'''+ ' and calendar='+DateToStr(date_temp);
    DataModule_head.Table_head.Open;

end;

procedure TForm_perday.btn_nextClick(Sender: TObject); //单击“明天”按钮
var
    date_temp:TdateTime;
begin
    if last_day=DayPerMonth(last_day,last_month) then //表示当前日期是本月最
后一天
    begin
        if last_month=12 then //如果是12月份，则执行下面的代码
        begin
            last_year:=last_year+1;

```

```

        last_month:=1;
    end
    else
        //不是12月份时执行的代码
        last_month:=last_month+1;

        last_day:=1;           //“明天”是下月的1号
    end
    else
        last_day:=last_day+1;

        date_temp:=EncodeDate(last_year,last_month,last_day);
        DataModule_head.Table_head.Filter:='username='+'''+form_head.DBEdit_user.text
        +'''+ ' and calendar='+DateToStr(date_temp);
        DataModule_head.Table_head.Open; //查询数据
    end;

    procedure TForm_perday.DB_E_PdateDblClick(Sender: TObject); //通过日历输入日期
    begin
        if DBE_Pdate.Text<>'' then
            begin
                DeCodeDate(DBE_Pdate.Field.Value,last_year,last_month,last_day);
                end;

                glocale_str:='head_oneday'; //设置全局变量，表示当前显示的是“一天工作安排”
                F_calendinput.Showmodal;
            end;

            procedure TForm_perday.DB_E_PdateExit(Sender: TObject); //该事件用于计算“星期”
            const
                day_no: array[1..7] of
                    string=(' (日)', ' (一)', ' (二)', ' (三)', ' (四)', ' (五)', ' (六) ');
            var
                week_no: integer;
            begin
                if DBE_Pdate.Text<>'' then
                    begin
                        DBE_Pname.text:=form_head.DBEdit_user.text ;
                        week_no:=DayOfWeek(DBE_Pdate.Field.Value);
                        DBE_Pweek.Field.Text:=day_no[week_no];
                    end ;
                end;
            end.

```

### 8.2.7 添加全局单元文件

我们以前讲述的实例中，每个单元文件都对应一个表单。但是有时不需要使用表单，

而只需要使用一些单元文件。这些单元文件可以定义一些全局变量、全局函数或过程等，可以通过在其他单元文件中使用 `uses` 语句调用该单元，因此就可以在这些单元文件中调用这些变量、函数或过程等。

在该项目中新添加一个单元文件的方法如下：选择 `File|New|Unit` 菜单，则可以新建一个单元文件，将该单元文件保存到保存项目文件的目录中，将其名字修改为 `unit head_public_unit`。

**注意：**实际上在建立一个项目时，应该提前将使用的单元文件规划好并建立共用单元文件。在前面的多个单元文件中已经调用了该单元文件，现在再对它作一些分析。

该单元文件完整的代码如下：

```
unit head_public_unit;

interface
uses
    SysUtils;

var
    last_year,last_month,last_day:word; //用于保存最后的日期
    globle_str,g_calend:string;        //作为全局变量

    Function LeapYear(Ayear:integer):Boolean;
    function DayPerMonth(Ayear,Amonth:integer):integer;
    procedure get_filter(ayear,amonth,aday:word);

implementation

uses head_main,head_gblogin,head_module; //引用其他单元

Function LeapYear(Ayear:integer):Boolean; //用于计算闰年，用于下面的函数中
begin
    Result:=(Ayear mod 4 =0) and ((Ayear mod 100<>0)
        or (Ayear mod 400 = 0));
end;

//可以不使用下面的函数，而直接使用Delphi提供的DayOfTheMonth
function DayPerMonth(Ayear,Amonth:integer):integer;
const
    DayInMonth:array[1..12] of Integer
    =(31,28,31,30,31,30,31,31,30,31,30,31);
begin
    Result:=DayInMonth[Amonth];
    if (Amonth=2) and LeapYear(Ayear) then
        inc(Result);
end;

procedure get_filter(ayear,amonth,aday:word); //根据年、月、日建立筛选条件
函数
```



```

var
    week_one, week_seven, i, day_no : integer;
    weekone_date, weekseven_date    : Tdatetime;
begin
    //#####以下计算本周一到周日的日期并作数据查询条件####
    if Dayofweek(EncodeDate(ayear, amonth, aday)) > 1 then
        day_no := Dayofweek(EncodeDate(ayear, amonth, aday)) - 1 //周1--周6
    else day_no := 7; //周日

    //以下计算周一的日期
    i := aday + 1 - day_no;
    if i > 0 then
        begin
            week_one := i;
            weekone_date := EncodeDate(ayear, amonth, week_one);
        end
    else
        begin
            if amonth < 1 then
                begin
                    week_one := DayPerMonth(ayear, (amonth - 1)) + i;
                    weekone_date := EncodeDate(ayear, (amonth - 1), week_one);
                end
            else
                begin
                    week_one := DayPerMonth(ayear - 1, 12) + i;
                    weekone_date := EncodeDate(ayear - 1, 12, week_one);
                end;
            end;

            //以下计算周日的日期
            i := aday + 7 - day_no; //i用来存放周日总天数
            if i <= DayPerMonth(ayear, amonth) then
                begin
                    week_seven := i;
                    weekseven_date := EncodeDate(ayear, amonth, week_seven);
                end
            else
                begin
                    week_seven := i - DayPerMonth(ayear, amonth);
                    if amonth < 12 then
                        weekseven_date := EncodeDate(ayear, (amonth + 1), week_seven)
                    else
                        weekseven_date := EncodeDate(ayear + 1, 1, week_seven);
                    end;

                    DataModule_head.Table_head.Close;
                    {下面的语句设置筛选条件, form_head.DBEdit_user.text 的前后使用'',
                    是因为其返回值为一个字符串, 但是该筛选条件中需要引用字符串。
                    ''表示一个单引号 (')。}

```

```

DataModule_head.Table_head.Filter:='username='+'''+form_head.DBEdit_use
r.text
+'''+ ' and (calendar>='+DateToStr(weekone_date)+' )'
      +' and (calendar<='+DateToStr(weekseven_date)+' )';
DataModule_head.Table_head.Filtered:=True;
DataModule_head.Table_head.Open;

end;

end.

```

注意：上面的共用单元文件中没有{\$R \*.DFM}语句，因为它没有对应表单。

### 8.2.8 完善项目文件

项目文件用于保存整个项目中表单的定义、创建等功能。另外，该项目文件中加入了处理运行参数的一些代码。在设计阶段，可以通过 Run|Parameters 菜单打开一个参数设置对话框，如图 8-26 所示。可以在 Parameters 一栏输入注册到数据库所使用的参数，前面是用户名，后面是口令，中间使用空格间隔。

在运行阶段，可以将参数放在该项目可执行文件的后面。

如果已经设置了运行参数，则不需要显示口令注册窗口，也就不需要创建该窗口了。

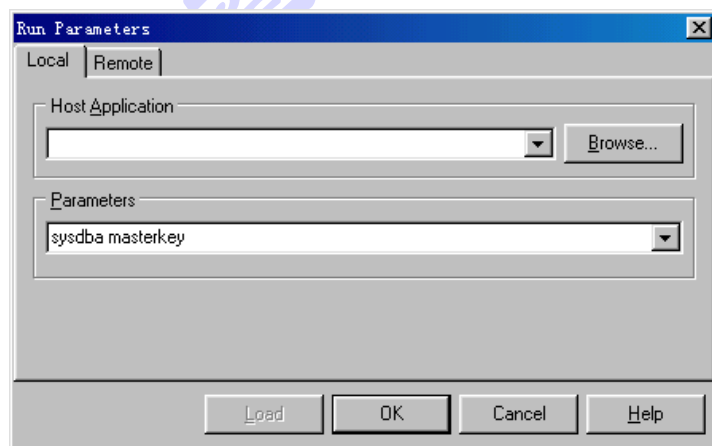


图 8-26 设置应用程序的运行参数

项目文件对应的代码如下：

```

..... //省略文件头
head_main in 'head_main.pas' {form_head},
head_module in 'head_module.pas' {DataModule_head: TDataModule},
head_gblogin in 'head_gblogin.pas' {frmlogin},
head_perday in 'head_perday.pas' {Form_perday},
head_public_unit in 'head_public_unit.pas',
head_calend_input in 'head_calend_input.pas' {F_calendinput};

{$R *.RES}

begin
  Application.Initialize;

```

```

Application.CreateForm(Tform_head, form_head);
Application.CreateForm(TDataModule_head, DataModule_head);
//下面的代码处理设置的运行参数
if ParamCount=2 then
begin
    DataModule_head.Databasel.Params.Clear;
    datamodule_head.Databasel.Params.Add('USER NAME='+paramstr(1));
    datamodule_head.databasel.params.add('PASSWORD='+paramstr(2));
    g_username:=paramstr(1);
    g_password:=paramstr(2);
    datamodule_head.databasel.connected:=true;
end
else //如果没有设置参数或参数不正确,则执行下面的代码
begin
    screen.Cursor:=crArrow;
    Application.CreateForm(Tfrmlogin, frmlogin);
    frmlogin.showmodal;
end;

Application.CreateForm(Tform_head, form_head);
Application.CreateForm(TForm_perday, Form_perday);
Application.CreateForm(TF_calendinput, F_calendinput);
Application.Run;
end.

```

### 8.2.9 运行应用程序

现在运行应用程序,首先会打开口令注册窗口,在该窗口中输入正确的用户名和口令,如图 8-27 所示。



图 8-27 在口令注册窗口输入用户名及口令

单击“确定”按钮,如果用户名及口令正确,则会打开应用程序的主表单,同时会查询本周的计划,如图 8-28 所示。此时可以输入新的数据,也可以通过下面的一排按钮查询数据。

如果单击“日历”按钮,则会打开日历窗口,如图 8-29 所示。通过选择一个合适的日期,并单击“确定”按钮,则可以查询任何一周的计划并显示在主界面中。

在输入数据时,如果双击日期字段,也会打开日历窗口,此时单击“确定”按钮后会输入数据。

如果通过“日历”按钮打开日历窗口后，在日期表单中双击鼠标，则会打开一天工作安排的窗口，并显示出最后选择的日期对应的计划，如图 8-30 所示。可通过该窗口下面的按钮查询前一天或后一天的工作安排，也可以通过该窗口插入、修改或删除数据。



图 8-28 打开应用程序主窗口时会查询本周的工作安排



图 8-29 该日历窗口可以查询数据、输入日期值或打开一天工作安排窗口



图 8-30 一天工作安排窗口

## 第9章 打印报表

在实际开发应用程序时，一般都离不开打印功能。Delphi 6 中提供了完善、易用的打印组件，这些组件都放置在 QReport 面板上，利用它们可以非常方便地实现各种报表的打印。在 Delphi 的 Demos 子目录中也提供了两个完整的实例 (...\\demos\\Quickrpt)，可以通过分析这两个实例学习使用 Quickrpt 开发打印。本章通过具体分析符合我们习惯的一个完整的打印例子，以便掌握使用 QReport 面板上的组件实现打印的方法。

### 9.1 QReport 面板组件介绍

QReport 面板提供的组件可以分类以下几种类型：

1. 模板类组件 这类组件作为快速报表的模板或容器，可以放置显示具体内容的组件，它们实际上是 QReport 组件面板最左边的 6 个组件，这些组件包括 QuickRep、QRSubDetail、QRStringBand、QRBand、QRChildBand、QRGroup 组件。其中 QuickRep 是必用组件，它必须插入到一个空白表单中，是建立快速报表的基础，该面板的所有都要放置在组件之上。如果没有在一个表单中插入该组件，则将无法建立快速报表。QRBand 也是必用组件，它要插入到 QuickRep 组件实例上面，该组件用于显示各种具体内容，内容的类型由其 BandType 属性决定。其他几个组件使用频率低一些，一般用于特定类型的报表。

2. 显示一般内容的组件 该类组件用于显示非数据库类型的数据或提供一些特定的功能。这类组件包括 QRLabel、QRExpr、QRSysData、QRMemo、QRExprMemo、QRRichText、QRShape、QRImage、QRChart 等。其中 QRExpr 组件既可以显示一般的数据，也可以显示数据库数据。

3. 显示数据库内容的组件 这类组件用于显示数据库中的数据，这些组件包括 QRDBText、QRExpr、QRDBRichText、QRDBImage 等。

4. 其他组件 还有一些组件实现其他功能，其中 QRCompositeReport 用于合并多个报表、QRPreview 用于预览报表、QRTextFilter、QRCSVFilter、QRHTMLFilter 等组件则用于输出报表的内容。这些组件使用较少。

下面分别说明各个组件的作用。

#### 9.1.1 模板类组件

**QuickRep** 该组件是基本的报表格式，它必须插入到一个表单 Form 上。它是一个可见的组件，它的形状由当前选择的打印纸的大小决定。

(1) QuickRep 的主要属性 其主要属性是 Dataset，该属性的作用如下：

- **Dateset** 使用该属性设置报表连接的数据集，该数据集可以是 TQuery、Table、TRemoteDataset 或其他 TCustomDataset 的子类，这样，通过一个细节显示区域 (detail band) 显示数据库中的记录。并从第一条记录开始显示。

(2) 上下文菜单 当在表单中插入一个报表时，如果报表没有与表单对齐，则可以用

鼠标拖拉该报表，使其左上角与表单的左上角重合。如果在该报表上面单击鼠标右键，则会弹出一个上下文菜单，其中有两个菜单比较重要，这两个菜单的作用如下：

- **Report settings** 选择该菜单后，则会弹出一个对话框，如图 9-1 所示。该面板实际上是报表的设置面板。

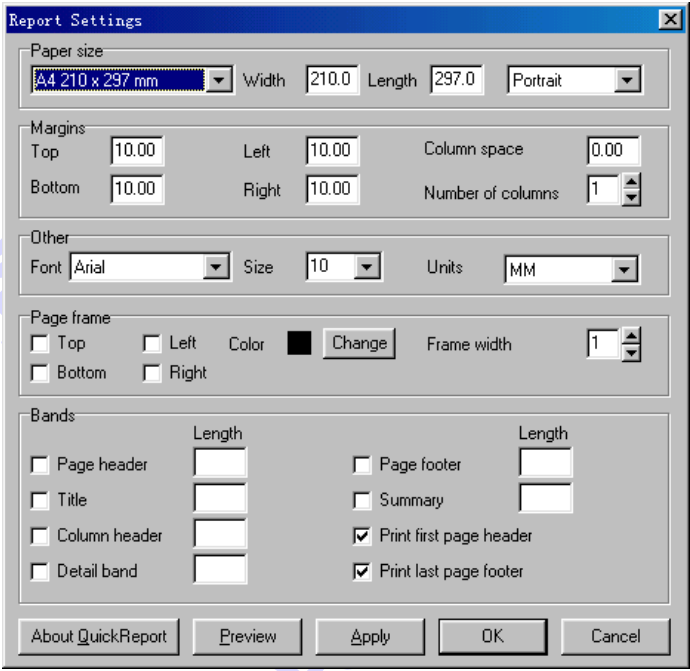


图 9-1 报表设置面板

该面板可以划分为 6 部分，这 6 部分的作用如下：

◇ **Paper size** 该部分设置打印页面的大小，可以从最左边的下拉框中选择合适的打印纸，默认情况下是 A4 打印纸；Width 和 Length 是当前选择的打印纸的宽度和长度；最右边的一个选项则是打印的方向，Portrait 表示纵向打印，而 Landscape 则表示横向打印。

◇ **Margins** 该部分设置打印的页边距和列边距，Top、Bottom、Left、Right 分别表示打印内容离打印纸上、下、左、右边界的距离；Column space 则表示打印内容列与列之间的间距；Number of columns 则表示一页纸中打印几栏。

◇ **Other** 该部分主要设置打印字体，Font 选项设置打印字体的类型；Size 设置打印字体的大小；Units 设置打印字体的单位。

◇ **Page frame** 该部分设置报表周围是否使用边框及边框的颜色和宽度；如果选择 Top、Bottom、Left、Right 中的任一选项或多个选项，则表示在页面对应的方向上显示边框；Color 选项设置边框的颜色，可以通过 Change 按钮设置；Frame width 设置边框的宽度，以像素为计量单位。

◇ **Bands** 该部分设置各个显示区域的宽度及打印效果。一个完整的打印页面可以划分为页头（Page header）、标题（Title）、各列内容的标题（Column header）、细节区域（Detail band）、页脚（Page footer）、汇总（Summary）等几部分，它们按照从上到下的顺序排列。如果选中一个选项，单击“Apply”或“OK”按钮后，则对应的区域类型就会添加到报表



中；同样，如果通过 QReport 组件面板上的 QRBand 组件在报表中插入一种报表区域类型时，则再打开该面板时也会在该部分选中该区域对应的选项。实际上这 6 种类型都是通过 QRBand 组件的 BandType 属性设置的。由以上分析可见，通过图 12-1 中的 Bands 部分选项插入不同的 QRBand 区域类型更简单。

另外，如果选择“Print first page header”或“Print last page footer”，则当打印多页内容时，只会打印第一个页面的页头和最后一个页面的页脚。

◇ 按钮 面板的底部是一排按钮，其中“About QuickReport”显示当前报表的情况说明；“Preview”则可以预览当前报表的打印效果；“Apply”会将该面板的设置选项作用到当前报表中；“OK”按钮会应用当前设置的选项并关闭该面板；“Cancel”则取消当前进行的设置并关闭该面板。

- Preview 该菜单非常有用，可以在编辑状态下随时查看报表的设计效果，其打开的面板如图 9-2 所示，该报表只设置了左右边框，而没有显示任何数据。

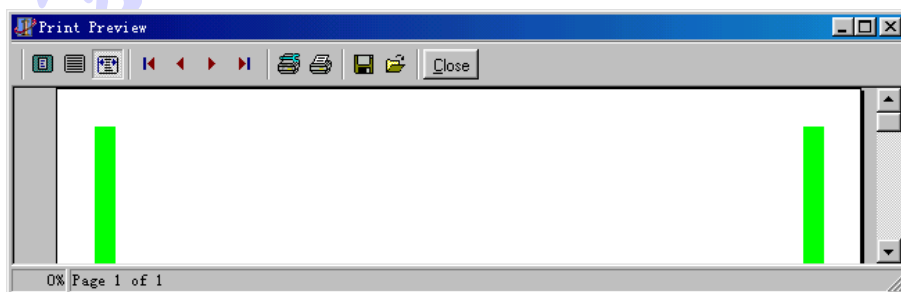


图 9-2 Preview 菜单打开的预览面板

2. QRSubDetail 该组件只有在建立一个主/从（master/detail）报表时才使用，主要用于显示从属表的数据。其主要的属性如下：

- DataSet 设置该组件显示的从属表对应的数据集。该数据集可以是一个 Table 组件，也可以是一个 Query 组件，这些数据集组件通过 Data Access 面板的相应组件添加。
- Master 该属性用于建立报表正确的主从关系。一般情况下该属性设置为 QuickRep，也可以设置为其他的报表控制组件。
- PrintIfEmpty 有时从属表中没有数据，此时我们希望在报表中不要显示该部分内容，则应将该属性设置为 True。
- FooterBand 如果在一个主从报表中希望在每次显示从属表的区域下面同时显示一些统计信息等，则可以通过 QRBand 组件添加一个注脚区域组件，则该属性中应设置该组件。同时，将 Bands 属性的子属性 HasFooter 属性设置为 True。注意此时 QRBand 组件中的 BandType 属性通常设置为 rbGroupFooter。
- HeaderBand 如果需要在显示每条从属表详细记录的区域前显示一些对从属表每条记录的说明或统计信息等，可以通过 QRBand 组件添加一个区域头组件，此时 QRBand 组件中的 BandType 属性通常设置为 rbGroupHeader。同时，将 Bands 属性的子属性 HasHeader 属性设置为 True。

3. QRStringBand 该组件的目的般用于显示一个字符串列表中的各行字符串内容，

一般不会连接到一个数据集。这是与其他组件最大的不同点。其主要属性有：

- **Items** 该属性是一个字符串的列表，可以在打开的编辑框中输入需要显示的字符串。
- **LinkBand** 如果要使另一个报表区域，比如注脚区域、汇总区域与当前的字符串区域显示在同一个页面上，则应在该属性中设置同时显示的区域。
- **Master** 一般设置为放置该组件的 QuickRep 组件。

4. **QRBand** 该组件是建立报表时最常用的，可以通过该组件实现报表中大部分区域的显示。这些不同的区域是通过其 **BandType** 属性值决定的。其主要的属性如下：

- **BandType** 该属性决定了显示报表区域的类型，共有如下可选值：**rbTitle**、**rbPageHeader**、**rbDetail**、**rbPageFooter**、**rbSummary**、**rbGroupHeader**、**rbGroupFooter**、**rbSubDetail**、**rbColumnHeader**、**rbOverlay**、**rbChild**。下面分别说明各种类型的作用：

◇ **rbTitle** 用于显示报表的标题，显示在页面表头的后面。如果报表有多页，则该区域只显示在报表的第一页中。该值是 **BandType** 属性的默认值。

◇ **rbPageHeader** 该选项设置页面的表头，显示在报表每个页面的最上面。可以通过 **Options** 属性来决定是否只显示在第一页或最后一页。

◇ **rbDetail** 该值是最常用的选项，用于显示一个数据集中的每一行或每一条记录。要显示基表中的记录，应在该区域放置一些 QRDB 开头的报表组件，比如可以放置一个 QRDBImage 来显示数据集中的图像。

◇ **rbPageFooter** 与 **rbPageHeader** 相反，用于显示页面的页脚，显示在报表每个页面的最下面。可以通过 **Options** 属性来决定是否只显示在第一页或最后一页。比如可以在页脚中显示日期、页号等信息。

◇ **rbSummary** 该选项显示汇总信息，一般在显示完所有的数据集记录或组合脚注后，在报表的最后显示该区域。

◇ **rbGroupHeader** 该选项用于显示 QRGroup 或 QRSubDetail 组件的头区域，在显示任何细节区域前显示该区域。

◇ **rbGroupFooter** 该选项用于显示 QRGroup 或 QRSubDetail 组件的脚注区域，在显示任何细节区域后显示该区域。

◇ **rbSubDetail** 该选项是为 QRSubDetail 组件保留的，不要手工设置该区域类型。

◇ **rbColumnHeader** 在一个多列的报表中，使用该区域显示各个页面的列标题。在一个单列的报表中，只在页头后打印一次。一般在该区域添加一些 QRLabel 等组件。

◇ **rbOverlay** 该选项主要用于报表版本向后的兼容性，所以不要使用该区域类型。

◇ **rbChild** 该选项是为 QRChildBand 组件保留的，所以不要手工设置该区域类型。

5. **QRChildBand** 该组件一般产生一个区域附加在另一个报表区域的下面。如果你希望一些控件依赖于其他控件的垂直展开时向下移动，则该组件特别有用。将要展开的控件放置到一个报表区域中，然后将其他要向下移动的控件放置到一个子报表组件上。可以通过其他组件的 **HasChild** 及 **LinkBand** 属性将一个子报表区域连接到另一个报表区域的下面。其主要的属性有：

- **ParentBand** 设置与子报表区域相关的报表区域，如果选择了一个合适的区域，则

该子区域总是与该区域同时显示，并且显示在其下面。

**6. QRGroup** 如果希望将一个报表中的数据按照一定的规则分组显示，则需要使用该组件。该组件一般当作一个分组的组头区域，有时也可以作为组脚区域。数据的分组主要是通过 Expression（表达式）属性的运算结果决定的。在进行多层分组的情况下，如果要改变分组的顺序，则可以在该组件上面单击鼠标右键，然后选择“Move Group up”，则该组就移到了上面。其主要的属性如下：

- **Expression** 该属性决定分组使用的表达式。比如要按照人员名字的字母顺序排列一个单位的所有人的基本情况，假设使用 name 表示人员名字字段，则可以在分组区域的该属性中输入如下表达式：copy(name,1,1)。其中 copy 是一个字符串函数，表示拷贝 name 字段的第一个字母。

要设置该属性，可以单击右边的按钮“...”，则打开的对话框如图 9-3 所示。



图 9-3 设置分组表达式

该对话框可以划分为三部分：上面的编辑框用于输入表达式；Insert at cursor position（插入光标位置）部分包括的多个按钮会在光标位置插入相应内容；底部的四个按钮是功能按钮，其中“Clear”用于清除，“Validate”用于检验表达式的有效性，“OK”按钮会关闭该面板并确认当前的表达式，“Cancel”会取消当前对话框中进行的各种操作。

下面重点说明第二部分各个按钮的作用。

◇ **Database field**（数据库字段） 该按钮用于在表达式中插入数据集的一个字段或多个字段，其打开的对话框如图 9-4 所示。对话框的左边列出了所有可以选择的数据集，右边的列表框中则是当前选择数据集对应的所有字段，可以选择需要的一个字段。比如当在右边一栏中选择了 Phone 字段时，由于其对应的数据集是 Customer，则会在表达式编辑框中输入 Customer.Phone 的内容。

◇ **Function** 可以通过该按钮在表达式编辑框中加入实现特定功能的函数，比如数据的累加、求平均值等。其打开的对话框如图 9-5 所示。该对话框左边一栏按照函数的功能进行了分类，而右边列表框中则是当前选择类别对应的所有函数。如果左边的类别中选择 All，则右边的列表框中会列出可以使用的所有函数。该对话框的下面显示了当前选择的函

数的用法及使用说明。比如，如果在 Category（类别）一栏选择了 Statistical（统计），然后在右边一栏选择了 SUM，则下面会显示出其语法为 SUM(<X>)，其中<X>是表示一个数字或一个数字表达式。

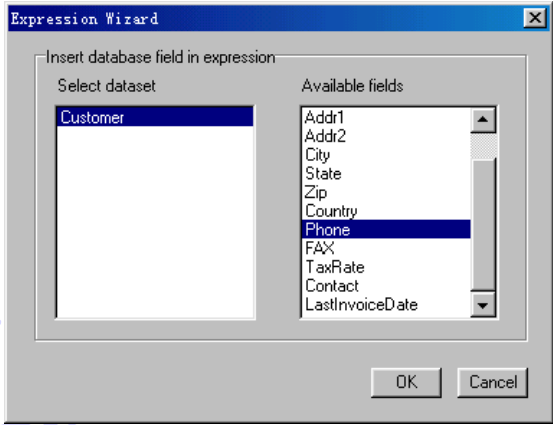


图 9-4 在表达式中插入数据库的字段

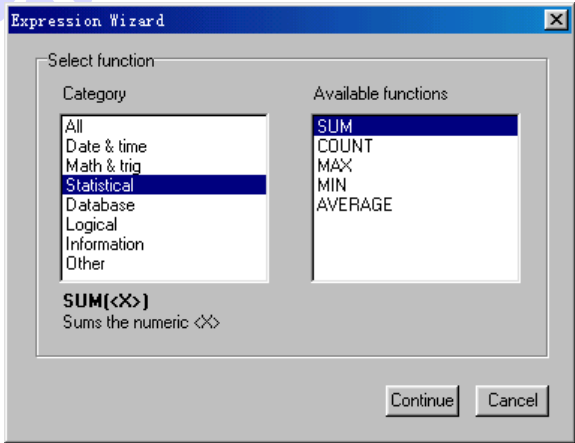


图 9-5 选择需要的函数

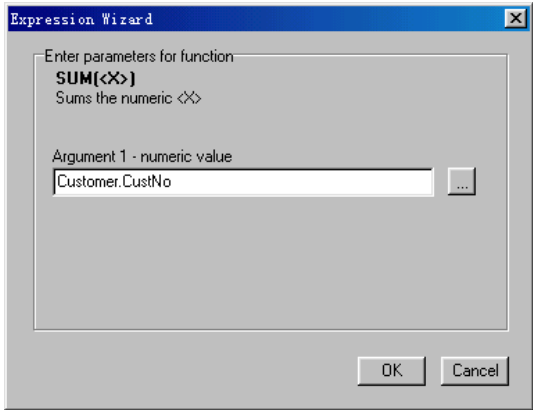


图 9-6 输入选择函数的参数

选择了合适的函数后，单击 Continue（继续）按钮，则会打开一个新的对话框，如图

9-6 所示,该对话框主要用于为函数设置合适的参数。如果参数要设置成一个表达式,则可以单击右边的按钮“...”,则又会打开类似于图 9-3 所示的编辑框,不过此时设置的表达式是作为当前选择函数的参数使用。比如通过“Database field”按钮设置当前的参数为 Customer.CustNo,则单击图 9-6 的“OK”按钮又会回到 9-3 所示的对话框中,并生成最终的表达式 SUM(Customer.CustNo)。

◇ Variable 该按钮会打开一个对话框,用来选择合适的变量,该对话框如图 9-7 所示。对话框中列出了已经定义的一些变量,要修改一个变量,可以选择下面的“Modify variables”(修改变量)按钮,则会打开一个新的对话框,然后可以将变量的默认值修改为需要的值即可。在一个表达式中可以直接使用一个变量,如果修改了变量的值,则表达式的值也会相应改变。

当然,如果需要,可以自己定义其他的变量,并设置合适的默认值。

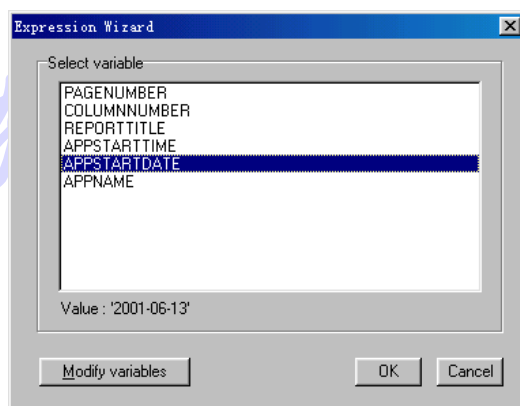


图 9-7 选择已经定义变量的对话框

◇ +、-、...And、Or 这一排按钮分别用于表达式的关系或逻辑运算,自左向右各个按钮进行的运算分别是:加法、减法、乘法、除法、等于、小于、大于、不等于、小于等于、大于等于、逻辑非、逻辑与、逻辑或。

### 9.1.2 显示一般内容的组件

1. QRLabel 该组件的作用是在一个区域中插入一个标签。其主要属性如下:

- Caption 设置标签的标题,可以将其修改为需要显示的标题的内容,比如一个数据库列的名字。
- Name 设置标签的名字,可以将其修改为容易识别或记忆的名字。

2. QRExpr 在报表区域中插入一个表达式,当运行报表时将显示该表达式的结果。

其主要属性如下:

- Expression 该属性设置用于设置一个表达式,其对话框如图 9-3 所示。表达式中最终的结果是以字符串的形式出现的。

3. QRSysData 该组件用于在一个报表区域中插入合适的系统数据,数据的格式由 Data 属性决定,Data 也是该组件的主要属性。

- Data 设置系统数据的格式,可以从下拉框中选择一个值。各个值的作用如下:

◇ qrsDate 在报表中显示当前系统的日期。格式如 2001-07-01。

- ◇ qrsDateTime 在报表中显示当前系统的日期和时间。格式如 2001-07-01 9:22.20。
- ◇ qrsDetailCount 显示 Detail 报表区域中显示数据集的记录数量。
- ◇ qrsDetailNo 显示 Detail 报表区域的编号。
- ◇ qrsPageNumber 显示当前报表页面的编号。
- ◇ qrsReportTitle 显示报表的标题。
- ◇ qrsTime 只显示系统的当前时间。

注意：以上系统日期或时间的显示格式有 Windows 控制面板的区域设置中的日期（短日期）和时间标签页的设置决定性。

4. QRMemo 该控件可以同时显示多行内容。其主要的属性如下：

- Lines 单击该属性右边的按钮，则可以打开一个文本编辑框，可以在该编辑框中输入多行文本内容。
- WordWrap 该属性决定到达 QRMemo 指定的宽度后（Width 属性决定）是否自动换行。设置为 True 则表示自动换行。

5. QRExprMemo 该组件实际上是 QRExpr 和 QRMemo 两个组件的混合体，它允许在多行的文本中使用表达式，但这些表达式必须使用大括号{}包括。在产生报表时会自动计算这些表达式的结果。其主要的属性与 QRMemo 的主要属性相同。比如，可以在 Lines 文本框中输入如下内容：Company : {CompanyName} Address : {Address1}，其中 {CompanyName} 和 {Address1} 是数据库一个基表中的两个字段，表达式的显示结果是这两个字段在相应记录上的值。

6. QRRichText 使用该组件可以显示 RichText 格式的多行文本内容。文本的内容取自 RichEdit 控件，可以通过 ParentRichEdit 属性获得其文本。其主要的属性如下：

- Lines 从 RichEdit 控件中读取的内容显示在该属性中。
- ParentRichEdit 设置一个 RichEdit 控件并从该控件中读取数据。RichEdit 控件来自 Win32 控件面板。

7. QRShape 通过该组件可以在报表的一个区域中插入合适的图形。比如可以加入一些直线将一些字段或记录间隔开。其主要的属性如下：

- Brush 设置一种刷子来填充图形的内部。
- Pen 设置一种画笔来绘制图形的轮廓。
- Shape 设置绘制图形的形状，可以从下拉框中选择一个值。各个值的作用如下：
  - ◇ qrsCircle 绘制一个圆形。
  - ◇ qrsHorLine 绘制一条水平线。可以将其 Height（高度）属性设置为 1。
  - ◇ qrsRectangle 绘制一个矩形。
  - ◇ qrsRightAndLeft 绘制一个矩形右边和左边的两条线。
  - ◇ qrsTopAndBottom 绘制一个矩形上面和下面的两条线。
  - ◇ qrsVertLine 绘制一条垂直线段，可以将其 Width（宽度）属性设置为 1。

8. QRImage 在报表中插入一幅图像。其主要的属性如下：

- Picture 设置显示的图像，可以单击右边的按钮打开图像选择对话框来设置合适的图像。
- Stretch 如果希望图像根据 QRImage 设定的区域伸缩，则将其设置为 True，这样



原来的图像会变形。一般情况下应保持为 False，不过如果显示区域较小，这样只会显示图像的一部分内容。

9. QRChart 显示一个直方图或一个饼图。其主要的属性如下：

- Chart 通过该属性右边的按钮可以打开如图 9-8 所示的对话框，一般情况下都要根据数据库的数据进行变化。可以在 Chart 标签页设置直方图的形状；在 Series 标签页设置其显示的格式等，并在 Data Source 面板设置显示的数据源中的字段。

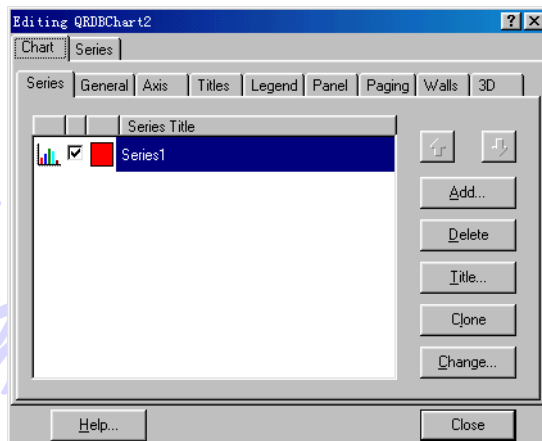


图 9-8 设置直方图或饼图的对话框

### 9.1.3 显示数据库内容的组件

以下几个组件主要用于显示数据库的数据，所以都要设置 DataSet 和 DataField 两个属性，其中 DataSet 设置使用的数据集，可以是一个 Table 或一个 Query 组件；而 DataField 属性用于设置显示的数据集字段。

1. QRDBText 显示文本类型的数据库字段。
2. QRDBRichText 显示 RichText 类型的多行数据库字段。
3. QRDBImage 显示图像类型的数据库字段。

### 9.1.4 其他组件

1. QRCompositeReport 该组件用于连接相关或不相关的报表。一般通过其 OnAddReports 事件连接相应的报表。

2. QRPreview 通过该组件可以建立一个自定义的运行时报表预览。在设计一个报表时，可以在 QuickRep 组件上面单击鼠标右键并选择 Preview 来预览报表，二者的作用类似。

3. QRTextFilter 在一个应用程序中，如果将该组件插入到一个表单中，则可以使报表输出到一个 ASCII 文本文件中，可以在保存文件的对话框中设置文件选项。

4. QRCSVFilter 该组件的作用与 QRTextFilter 组件的作用类似，只是它使报表输出到一个使用逗号间隔的文本文件中，可以在保存文件的对话框中设置文件选项。

5. QRHTMLFilter 该组件也是用于输出报表内容，不过它是以 HTML 文档的形式出现，可以通过 Web 浏览器浏览该页面。要使用该组件，也要将其放置到一个应用程序的表单中。

## 9.2 报表实例

下面通过一个具体的实例讲述如何通过 QReport 面板上的各个组件来建立一个可以打印的报表，该报表是一个主从报表，即每显示一条主表的记录，会同时显示与之对应的多条从属表的记录。该实例使用的数据库是 Microsoft Access 数据库。

### 9.2.1 建立基表

下面首先建立两个基表。主表是一所大学的学生名单，从属表则是各个学生的成绩单。打开 Microsoft Access 数据库应用程序，在“新建数据库”的几个选项中选择“空 Access 数据库”，然后单击确定按钮。如果希望使用一个已经建立的数据库，则可以选择下面的“打开已有文件”的选项。如果是新建一个数据库，则在接下来打开的保存新建文件数据库对话框“文件名”一栏输入一个数据库的名字，比如 test\_db，并选择一个保存该数据库的目录，然后单击“创建”按钮，则会创建一个新的数据库，并且打开数据库的管理界面，如图 9-9 所示，此时可以在该界面上建立新的基表结构并输入相应的记录，或进行其他操作。

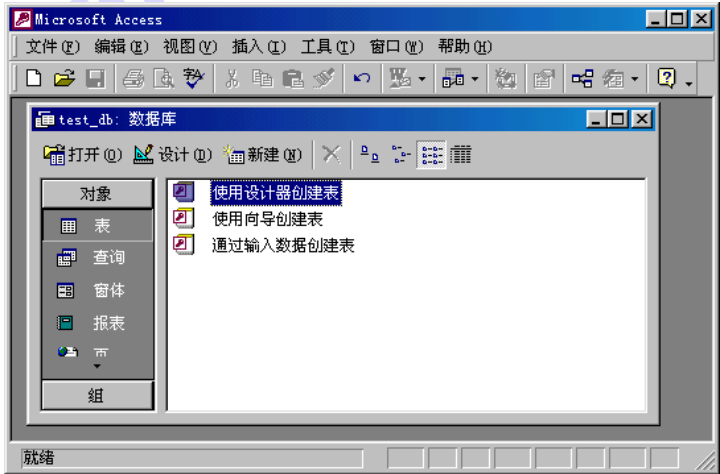


图 9-9 Access 数据库的设计及管理界面

#### 1. 建立主表 下面首先建立主表。

在 Access 数据库的管理面板中选择“使用设计器创建表”，然后上面双击鼠标，或者单击工具面板上的“设计”按钮，则可以给基表结构创建界面，然后可以输入该基表中各个字段的名称及数据类型等，并设置其他的参数。其设计界面如图 9-10 所示。

一个基表中一定要有一个字段设为主键，它用于确定一条记录的唯一性，即所有主键的值在基表中都必须是唯一的。设置主键的方法很简单，选择“编号”字段，然后单击工具条上的主键图标（类似一把钥匙）。设计完毕后，单击基表设计界面右上角的关闭按钮（即右上角的第二个“×”按钮）关闭设计面板，此时需要输入基表的名字，由于该基表包含的内容是所有学生的名单列表，所以可以将其命名为 name\_list。到此主表结构设计完毕。

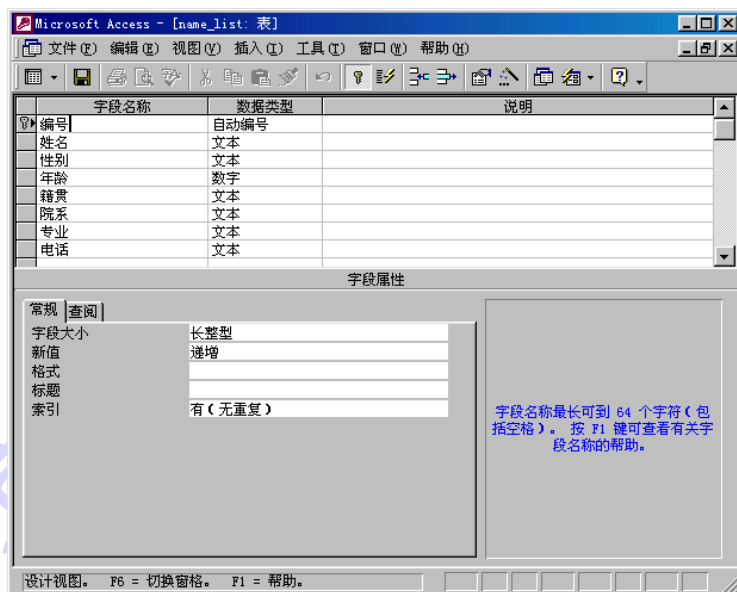


图 9-10 主表（学生名单列表）name\_list 的设计界面

2. 建立从属表 按照同样的方法，可以建立从属表，该表用于储存学生各门功课的成绩，所以可以将其名字设置为 **course\_list**，该基表结构的设计界面如图 9-11 所示。该基表中的“课程编号”被设置为主键。注意，该课程编号是对所有学生的所有课程统一编号，及每条记录有一个单独的编号。另外，该基表中有一个字段为“学员编号”，它与主表 name\_list 中的“编号”相对应，用于建立主从表的关联关系。

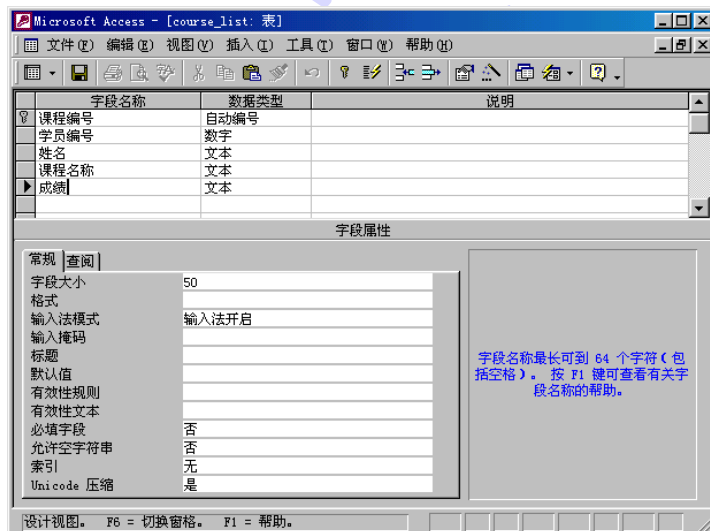


图 9-11 从属基表（学生课程成绩）course\_list 的设计界面

3. 建立主从基表的关系 要使主从基表相互作用，比如要删除一条主表记录，则应相应删除与之对应的从属表记录，此时就需要建立两个基表的关系。由于一名学生有多门课程的成绩，所以主从表之间的关系是一对多的关系。Access 提供了相应的菜单可以很方便地实现该功能。

(1) 添加建立关系的基表到关系面板 选择“工具/关系”菜单，然后在打开的基表列表面板中选择 name\_list，单击“添加”按钮，将该基表的名字添加到关系面板中。按照同样的方法将基表 course\_list 也添加到关系面板中。此时 Access 中会显示出“关系”系统菜单。

(2) 编辑关系 选择“关系/编辑关系”菜单，则会打开编辑关系面板，然后单击“新建”按钮，则在“左表名称”下拉框中选择 name\_list，在“右表名称”下拉框中选择 course\_list，同时分别在“左列名称”和“右列名称”中选择“编号”和“学员编号”。关闭该面板，则会返回到“编辑关系”面板。

在下面选择“实施参照完整性”选项，然后再选择下面的“级联更新相关字段”和“级联删除相关记录”，这样就可以建立两个基表“一对多”的关系类型。以上所有选项设置完毕后，“编辑关系”面板的显示如图 9-12 所示。

另外，如果要使主从表建立完整的对应关系，还需要设置它们的联接类型。设置方法如下：单击“联接类型”按钮，则可以打开联接属性，选择第一个选项，即主从表的关联只包含两个表联接字段相等处的行。该面板的显示如图 9-13 所示。

关闭“联接属性”和“编辑关系”面板后，则会返回到“关系”面板，如图 9-14 所示。



图 9-12 “编辑关系”设置面板

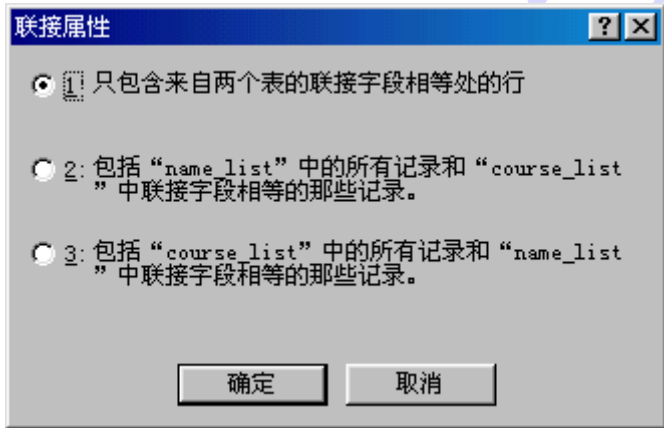


图 9-13 设置两个基表的联接属性面板

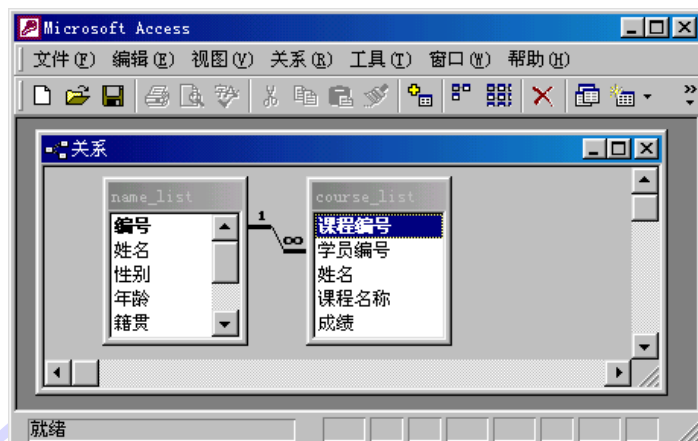


图 9-14 基表关系管理面板

从图 9-14 中的面板可以看出，name\_list 表的“编号”字段与 course\_list 表的“学员编号”字段建立了一对多的关系，用“∞”表示从属表的多条记录对应主表的一条记录。

4. 为主从基表输入数据 在基表中输入数据很简单。在数据库管理面板（见图 9-9）中双击一个基表的名字，比如 name\_list，则会打开基表的数据输入界面，然后按照各个字段的数据类型输入相应的数据即可。主表的数据输入界面如图 9-15 所示。

从属表 course\_list 中数据的输入有两种方法：一种是与输入主表数据使用相同的方法，其输入界面如图 9-16 所示，这种方法的缺点是需要提前确定从属表中各条记录的数据与主表中数据的对应关系；另一种方法是在主表的输入界面中输入从属表的数据，因为一旦建立了主从表的关联关系，则在主表每条记录的左边一列会显示出一个加号“+”，单击它则会打开与当前记录对应的从属表的数据输入界面，可以直接输入与当前记录对应的从属表的数据。

由于这是一个测试性质的实例，所以在主表 name\_list 中我们只输入了 6 条记录。注意，在输入数据时，由于“编号”字段的数据类型是“自动编号”，所以不要输入或修改该字段的数据。同样，在为从属表 course\_list 输入数据时，也不要为“课程编号”输入数据。



图 9-15 主表 name\_list 的数据录入界面

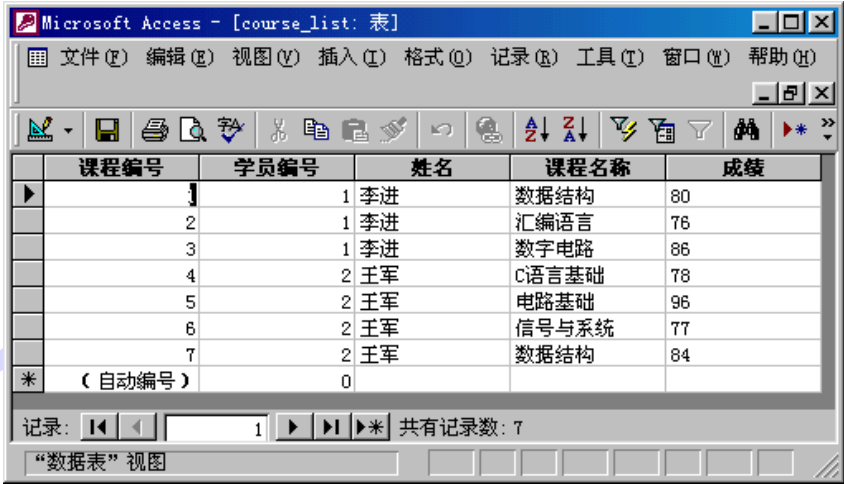


图 9-16 从属表 course\_list 的数据录入界面

9.2.2 建立应用程序

下面建立应用程序，该程序共有三个表单组成：一个表单是主表单，用来显示学生的花名册；第二个表单用于建立学生花名册的报表；第三个表单用于建立学生成绩列表的表单。

该实例的实现步骤如下：

- 1. 使用“File/New Application”菜单新建立一个程序，再使用“File/New Form”新建两个表单，然后将整个工程保存为 print\_test.dpr，并使用默认的名字保存三个单元文件。
- 2. 设计主表单 主表单主要用于显示学生的花名册，用户可对该表的数据进行增、删、查、改的操作。同时在该表单的下面需要添加四个按钮，分别用于预览和打印主表和从属表的报表。

（1）添加组件 打开 Data Access 组件面板，然后 DataSource 组件，将起添加到表单 Form1 的顶部；再选择 Table 组件，分别在 Form1 的顶部添加两个基表组件 Table1、Table2。

打开 DataControls 面板，然后在 Form1 的中间添加一个 DBGrid 组件，用于显示数据库的数据。再选择 DBNavigator 组件，将其添加到 Form1 的底部。

选择 Standard 面板的标签组件，在 Form1 的顶部放置一个标签；再选择 Additional 组件面板，选择 BitBtn 按钮组件，然后在 Form1 底部的左右两边分别放置 4 个按钮。

以上组件的名称都使用默认设置。

（2）设置组件属性 各个组件属性的设置值及作用，见表 9-1。

表 9-1 设置各个基本组件的属性

组件名称	属性	设置值	说明
DataSource1	DataSet	Table1	主表 name_list 对应的数据源
Table1	Active	True	显示基表 name_list 的数据



(续表)

组件名称	属性	设置值	说明
	DatabaseName	new_access	该数据库名字是在 BDE 管理器中建立的对应 test_db 数据库的数据库别名
	IndexFieldName	编号	索引字段的名字
	TableName	Name_list	主表的名字
Table2	Active	True	显示基表 course_list 的数据
	DatabaseName	new_access	该数据库名字是在 BDE 管理器中建立的对应 test_db 数据库的数据库别名
	IndexFieldName	学员编号	索引字段的名字
	MasterFields	编号	建立从属表“学员编号”与主表“编号”字段的对应关系
	MasterSource	DataSource1	将主表对应的数据源作为从属表的主数据源
	TableName	course_list	从属表的名字
DBGrid1	DataSource	DataSource1	设置对应主表的数据源
	Height	210	设置高度
	Width	680	设置宽度
DBNavigator1	DataSource	DataSource1	设置对应主表的数据源
	Height	33	设置高度
	Width	330	设置宽度
Label1	Caption	学生花名册	设置 Form1 的标题
	Font	隶书/粗体/一号/蓝色	设置标题字体
	Top	32	设置标题顶部位置
BitBtn1	Caption	预览名单	预览主表的按钮的标题
	Height	33	设置高度
	Width	80	设置宽度
BitBtn2	Caption	打印名单	打印主表的按钮的标题
	Height	33	设置高度
	Width	80	设置宽度
BitBtn3	Caption	预览成绩	预览从属表的按钮的标题
	Height	33	设置高度
	Width	80	设置宽度
BitBtn4	Caption	打印成绩	打印从属表的按钮的标题
	Height	33	设置高度
	Width	80	设置宽度

以上所有组件的属性都设置完毕后，最终的设计界面如图 9-17 所示。

图 9-17 设计完成的 Form1 主界面

9.2.3 建立报表

1. 建立学生花名册的报表 我们在 Form2 的基础上建立该报表。建立步骤如下：

（1）插入报表区域组件 首先在 Form2 中插入报表区域组件。

打开 QReport 组件面板，选择 QuickRep 组件，将其插入到 Form2 中，并使其左上角与 Form2 的左上角对齐，该组件使用默认的名字 QuickRep1。

选择 QRBand 组件，然后在 QuickRep1 组件连续插入 5 个 QRBand 组件，它们的名字依次为 QRBand1、QRBand2、QRBand3、QRBand4、QRBand5，然后分别将这 5 个组件的 BandType 属性依次修改为 rbPageHeader、rbTitle、rbColumnHeader、rbDetail、rbPageFooter，即分别用于显示页头、报表标题、字段名称、字段内容、页脚等区域。

该报表中各个区域组件的属性见表 9-2。

表 9-2 设置各个报表区域组件的属性

组件名称	属性	设置值	说明
QuickRep1	DataSet	Form1.Table1	设置报表对应的数据源
QRBand1	BandType	rbPageHeader	用于显示显示页头的区域
	Height	40	
QRBand2	BandType	rbTitle	用于显示报表标题的区域
	Height	80	
QRBand3	BandType	rbColumnHeader	用于显示字段名称的区域
	Height	45	
QRBand4	BandType	rbDetail	用于显示字段的详细内容
	Height	40	
QRBand5	BandType	rbPageFooter	显示页脚区域
	Height	72	

（2）插入显示具体内容的组件 下面在各个区域中插入显示具体内容的组件。

1) Page Header 区域 在该区域中显示报表的制作时间, 该时间使用当时的系统时间。选择 QReport 组件面板的 QRLabel 组件, 在页头区域的右边插入该组件作为标签, 将其 Caption 属性设置为“制表时间:”; 再选择 QRSysData 组件, 然后在标签的右边插入该组件, 将其 Data 属性设置为 qrsDate。

2) Title 区域 该区域用于显示报表的标题。选择 QReport 组件面板的 QRLabel 组件, 在报表标题区域的中间插入该组件作为标题, 将其 Caption 属性设置为“学生花名册”, 将 Font 属性设置为“隶书/规则/一号/蓝色”。

3) Column Header 区域 该区域用于显示主表字段的标题。选择 QReport 组件面板的 QRLabel 组件, 在字段头区域的插入 8 个该组件作为字段标题, 各个标题之间保持一定的间隔。将这些组件的 Caption 属性依次修改为“编号”、“姓名”、“性别”、“年龄”、“籍贯”、“院系”、“专业”、“电话”。

选择 QReport 组件面板的 QRShape 组件, 在 Column Header 区域中插入一个该组件, 然后将其 Height 属性设置为 40 (即与所在区域高度一样), Top 及 Left 属性都设置为 0, Width 属性设置为 723, Shape 属性设置为 qrsRectangle, 即表示是一个矩形, 用来作为字段标题的边框。再插入第二个 QRShape 组件, 将其 Shape 属性设置为 qrsVertLine, Width 设置为 1, 即表示插入了一条垂直线段, 并将其放置到“编号”和“姓名”字段的中间, 以便区分两个字段, 将其 Top 属性设置为 0; 按照同样方法, 分别在其他字段标题之间插入一条间隔线。

4) Detail 区域 该区域用于显示基表 name\_list 中字段的具体内容。这些字段都可以通过 QRDBText 组件显示。

首先在 Detail 区域中插入第一个 QRDBText 组件, 将其 DataSet 属性设置为 Form1.Table1, 将其 DataField 属性设置为“编号”; 按照同样方法, 再插入 7 个 QRDBText 组件, 将它们的 DataSet 属性都设置为 Form1.Table1, DataField 属性依次设置为“姓名”、“性别”、“年龄”、“籍贯”、“院系”、“专业”、“电话”。

注意: 如果只设置了 Detail 区域中各个数据库组件 (比如 QRDBText) 的 DataSet 和 DataField 属性, 而没有设置 QuickRep 组件的 DataSet 属性, 则只会显示一条记录。

5) Page Footer 区域 我们希望在区域显示报表的页号, 则可以直接插入 QRSysData 组件, 并将该组件的 Data 属性设置为 qrsPageNumber 即可。

以上所有组件的属性设置完毕后, “学生花名册” 报表页面的显示如图 9-18 所示。

2. 建立学生成绩的报表 建立学生成绩报表与建立学生花名册报表的方法相似, 只是该报表同时显示主从表的记录, 所以需要使用不同的报表组件。下面我们在 Form3 的基础上建立该报表。建立步骤如下:

(1) 插入报表区域组件 首先在 Form3 中插入报表区域组件。

打开 QReport 组件面板, 选择 QuickRep 组件, 将其插入到 Form3 中, 并使其左上角与 Form3 的左上角对齐, 该组件使用默认的名字 QuickRep2。

选择 QRBand 组件, 然后在 QuickRep2 组件连续插入 3 个 QRBand 组件, 它们的名字依次为 QRBand1、QRBand2、QRBand3, 然后分别将这 3 个组件的 BandType 属性依次修改为 rbTitle、rbColumnHeader、rbDetail, 分别用于显示主表标题、字段名称、字段内容等。



图 9-18 显示“学生花名册”的报表编辑界面

选择 QRSubDetail 组件，然后在 QRBand3（Detail 组件）的下面插入该组件，并使用默认的名字 QRSubDetail1。该区域用于显示从属表与主表当前记录关联的数据。

选择 QRBand 组件，然后在 QRSubDetail1 组件下面插入第 4 个 QRBand 组件，其名字为 QRBand4，并将其 BandType 属性修改为 rbGroupFooter。

该报表中各个区域组件的属性见表 9-3。

表 9-3 设置各个报表区域组件的属性

组件名称	属性	设置值	说明
QuickRep2	DataSet	Form1.Table1	设置报表对应的数据源为主表
QRBand1	BandType	rbTitle	用于显示报表标题的区域
	Height	67	
QRBand2	BandType	rbColumnHeader	用于显示字段名称的区域
	Height	27	
QRBand3	BandType	rbDetail	用于显示字段的详细内容
	Height	59	
QRSubDetail1	DataSet	Form1.Table2	设置该区域显示从属表数据
	FooterBand	QRBand4	设置该区域关联的页脚为 Group Footer 类型的区域
	PrintIfEmpty	True	如果从属表没有数据，则不显示该区域内容
QRBand4	BandType	rbGroupFooter	使用页脚区域显示记录的统计信息
	Height	24	

（2）插入显示具体内容的组件 下面在各个区域中插入显示具体内容的组件。

1) Title 区域 该区域用于显示报表的标题。选择 QReport 组件面板的 QRLabel 组件，在报表标题区域的中间插入该组件作为标题，将其 Caption 属性设置为“学员成绩列表”，将 Font 属性设置为“隶书/粗体/二号/褐红色”。

2) Column Header 区域 该区域用于显示主表部分字段的标题。选择 QReport 组件面

板的 QRLabel 组件，在字段头区域的插入 4 个该组件作为字段标题，各个标题之间保持一定的间隔。将这些组件的 Caption 属性依次修改为“学生编号”、“姓名”、“院系”、“专业”。

3) Detail 区域 该区域用于显示基表 name\_list 中字段的具体内容。这些字段都可以通过 QRDBText 组件显示。

首先在 Detail 区域中插入第一个 QRDBText 组件，将其 DataSet 属性设置为 Form1.Table1，将其 DataField 属性设置为“编号”；按照同样方法，再插入 3 个 QRDBText 组件，将它们的 DataSet 属性都设置为 Form1.Table1，DataField 属性依次设置为“姓名”、“院系”、“专业”。

由于该报表用于显示主从表的数据，所以还需要在该区域中设置从属表字段的标题。

按照插入主表字段标题的方法，在 Detail 区域的下半部分插入三个 QRLabel 组件，并将其 Caption 属性分别设置为“课程编号”、“课程名称”、“成绩”。

为了使不同的记录及从属表字段标题与主表的字段内容之间有所间隔，使用 QRShape 组件分别在 Detail 区域的顶部及中间插入两个该组件，并将它们的 Shape 属性都设置为 qrsHorLine，将 Pen/Style 属性分别设置为 psSolid、psDot，Width 属性分别设置为 710 和 345。

4) Sub Detail 区域 该区域用于显示从属表中与当前主表记录对应的内容。使用 QRDBText 组件分别在该区域中插入三个组件，将它们的 DataSet 属性都设置为 Form1.Table2，DataField 属性分别设置为“课程编号”、“课程名称”、“成绩”。

5) Group Footer 区域 该区域用于显示从属表记录的数量。选择 QRLabel 组件，在该区域中插入一个 QRLabel8 组件，将其 Caption 属性设置为“记录数量是：”；再选择 QRExpr 组件，在该区域中插入一个 QRExpr1 组件，将其 Expression 属性设置为 Count，Master 属性设置为 QRSubDetail1。

以上所有组件属性设置完毕后，Form3 的编辑界面如图 9-19 所示。

图 9-19 显示“学生成绩列表”的报表编辑界面

1. 编辑代码 代码的编辑可以分为如下几部分：

(1) 单元文件间的相互引用 由于 QuickRep1、QuickRep2 两个报表都要用到 Form1 中的 Table1 和 Table2 及数据集 DataSet1 等几个组件，所以 Unit2 和 Unit3 中要使用 uses 语句来引用 Unit1，为了与引用标准库单元的 uses 语句区别，该语句一般放在 {\$R \*.DFM} 语句的后面，该语句为：

```
uses unit1;    //引用unit1单元
```

同样，为了是 Form1 中的组件使用 Form2 和 Form3 中的 QuickRep1 和 QuickRep2，在 Unit1 中也应该加入如下语句：

```
uses unit2,unit3;    //引用unit2和unit3单元
```

(2)Form1 中四个按钮触发的事件 由于 Form1 中四个按钮分别用于预览和打印报表 QuickRep1 和 QuickRep2，可以通过 QuickRep 组件的 Preview 和 Print 两种方法实现这些功能，并使用四个按钮的 OnClick 事件。

四个按钮的完整代码如下：

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Form2.QuickRep1.preview;    //通过第一个按钮预览Form2中的QuickRep1报表
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    Form2.QuickRep1.print;    //通过第二个按钮打印Form2中的QuickRep1报表
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
    Form3.QuickRep2.preview;    //通过第三个按钮预览Form3中的QuickRep2报表
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
begin
    Form3.QuickRep2.print;    //通过第四个按钮打印Form3中的QuickRep2报表
end;
```

(3)控制 QuickRep2 中从属表字段标题的显示 在显示从属表的数据时，我们希望记录为空则不应显示标题，这可以通过 QRBand3 的 BeforePrint 事件实现，设置方法如下：

选择 QRBand3（即 Detail 区域），按 F11 键打开对象观察器，然后在 BeforePrint 事件右边一栏的空白位置上双击鼠标，则会产生一个 QRBand3BeforePrint 事件，此时对象观察器的显示如图 9-20 所示。在该事件中输入合适的代码，这些代码的作用是：如果从属表没有记录，则不显示 Detail 区域中“课程编号”、“课程名称”、“成绩”三个字段标题及标题上面的间隔线，同时将 Detail 区域的高度缩小。

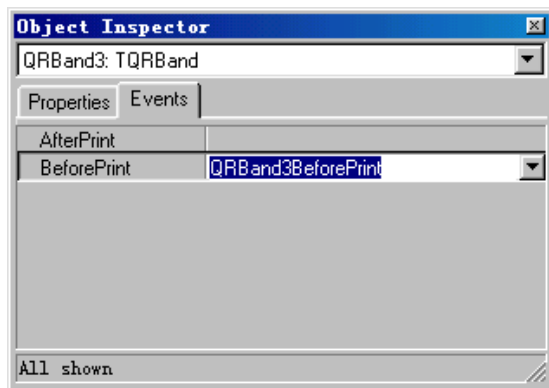


图 9-20 QRBand3 的事件设置面板



该事件的完整代码如下：

```
procedure TForm3.QRBand3BeforePrint(Sender: TQRCustomBand; var PrintBand:
Boolean);
begin
    if form1.Table2.RecordCount=0 then    //如果从属表中记录为空，则执行下面的代
    码
    begin
        QRLabel1.Enabled:=false;        //不显示“课程编号”字段标题
        QRLabel2.Enabled:=false;        //不显示“课程名称”字段标题
        QRLabel3.Enabled:=false;        //不显示“成绩”字段标题
        QRShape3.Enabled:=false;        //不显示从属表字段标题上面的间隔线
        QRBand2.Height:=42;            //降低Detail区域的高度
    end;
end;
```

#### 9.2.4 运行打印程序

最后按 F9 键运行该程序，在数据库用户名和口令输入窗口中（见图 9-21）不要输入任何内容，直接单击“OK”按钮即可，此时会显示出该应用程序的主界面，如图 9-22 所示。单击该界面的“预览名单”、“预览成绩”按钮，则打开的报表分别如图 9-23 及图 9-24 所示。

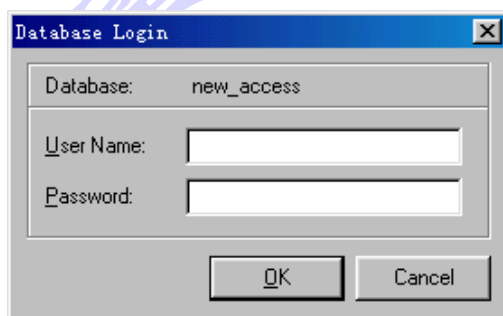


图 9-21 数据库认证窗口



图 9-22 应用程序的主界面

Print Preview

制表时间: 2001-05-19

**学生花名册**

编号	姓 名	性 别	年 龄	籍 贯	院 系	专 业	电 话
1	李进	男	20	北京	计算机系	计算机硬件	64333322
2	王军	男	22	天津	计算机系	计算机软件	86443323
3	孙雨	女	21	上海	电子系	信息处理	65334543
4	贺映	女	23	辽宁	电子系	通信系统	65344373
5	张新	男	21	陕西	数学系	系统工程	83220389
6	赵明	男	20	四川	物理系	基础理论	83223802

Page 1 of 1

图 9-23 主表报表的预览界面

Print Preview

**学员成绩列表**

学生编号	姓名	院系	专业
1	李进	计算机系	计算机硬件
3		数字电路	86
2		汇编语言	76
1		数据结构	80
记录数量是: 3			
2	王军	计算机系	计算机软件
7		数据结构	84
6		信号与系统	77
5		电路基础	96
4		C语言基础	78
记录数量是: 7			
3	孙雨	电子系	信息处理
4	贺映	电子系	通信系统
5	张新	数学系	系统工程

Page 1 of 1

图 9-24 从属表报表的预览界面

## 第 10 章 Decision Cube 面板与数据分析

有时一些应用程序需要对数据库的数据进行多维的分析，比如求一些字段数据的和、平均值及字段的数量等，此时就可以使用 Decision Cube 面板组件实现这些功能，这些组件可以建立交叉表格化的数据，并且可以进行多层次的汇总、求平均值等操作，这就为用户提供了一种非常直观的数据分析工具，从而帮助用户做出最终的决策。

该面板包括如下组件：

- TDecisionCube 一个多维的数据存储组件，用于连接 TDecisionQuery 和 TDecisionSource 组件。
- TDecisionQuery 继承自 TQuery，可以实现与其类似的功能，它们所具有的属性、方法及事件也十分相似，该组件用于定义 TDecisionCube 组件使用的数据。
- TDecisionSource 其作用类似于 TDataSource 组件，用于连接 TDecisionCube 和 TDecisionGrid 或 TDecisionGraph 组件。
- TdecisionPivot 相当于 TDBNavigator 组件，用于多维数据的导航。
- TDecisionGrid 以表格的形式显示一维或多维的数据，其作用类似于 TDBGrid。
- TDecisionGraph 以动态图形的方式显示来自一个决策表格的字段，当维数改变时，该组件显示的图形也会改变。

下面分别说明这些组件的作用。

### 10.1 Decision Cube 面板

#### 10.1.1 TDecisionCube 组件

TDecisionCube 组件使用多维表格的形式分析来自一个数据集的数据，在该表格中的每一维都对应数据集的一个字段。该组件可以从任何数据集中提取数据，但是与 TDecisionQuery 组件结合却效果最好。

一般情况下，TDecisionCube 组件使用的汇总值由数据集提供，但是，计算平均值则可以通过 TDecisionCube 组件内部实现，内部计算平均值的优点是，可以进行部分数据的汇总，并且这些数据可以被 TDecisionGrid 组件正确地处理。

类似其他数据库组件，可以将 TDecisionCube 组件放置在表单上，也可以放置在一个数据模块中，该组件需要通过 TDecisionSource 组件，才能向 TDecisiongrid 和 TDecisiongraph 对象提供数据。

##### 1. TDecisionCube 组件主要的属性

- DataSet 设置该组件使用的数据集，该数据集提供数据表格中使用的维数和汇总数据。通常使用的数据集是 TDecisionQuery 组件。
- DimensionMap 用于决定表格化数据的维数及汇总，维数地图的单独的像可以表示 DataSet 属性指定字段或来自那些字段的计算字段，比如平均值等。该属性提供

了这些字段的信息，包括字段的名称和在多维数组中的使用等。

单击 DimensionMap 属性右边的按钮，可以打开 Dimension Cube 编辑器，该编辑器有两个标签页，它们的意义如下：

◇ Dimension Settings 该标签页用来设置其对应数据集中使用的字段的一些属性。该面板如图 10-1 所示，该标签页的左边对应本章第一个实例中的 4 个字段，该标签页的左边是当前选择的字段（Available Fields 列表中）对应的一些属性，比如显示在界面上的名字、维数的类型、是否激活 Type 中设置的类型、使用的格式、数据分组方式及初始值等。

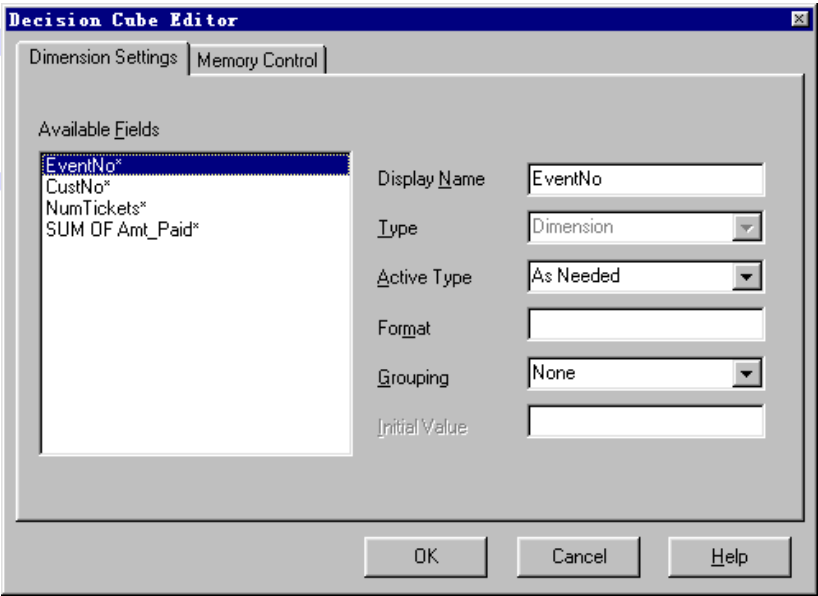


图 10-1 设置一些字段属性的 Dimension Settings 标签页

◇ Memory Control 标签页 该标签页用于设置 Decision Cube 可以使用的最大内存。由于 TDecisionCube 组件建立的应用程序会根据该组件维数及求和运算的增加，进行的数据计算也会成级数倍的增长，所以不但会消耗大量的内存，并且也会使用很长的时间进行计算，这会使应用程序的运行十分缓慢，所以适当限制显示的维数及求和运算的数量，会简化应用程序。该标签页如图 10-2 所示。

○ Cube Maximums 包括 4 行 3 列内容，其中第一行（Maximum）用于设置统计信息的最大维数（Dimensions）、汇总字段（Summaries）的数量及单元格（Cells）的数量，单元格 Cells 编辑框的设置值如果为 0，则表示根据维数和汇总数决定单元格的数目。

第二行（Current）表示当前显示的表格维数、汇总字段的数量及使用的单元格数量。

第三行（Active+Needed）表示全部显示统计信息时需要的表格维数、汇总字段的数量及使用的单元格数量。

第四行（Active）表示当前活动的维数及汇总字段。

○ Designer Data Options 该部分的 4 个单选项决定设计时使用的数据选项，其中 Display Dimension Names 表示只显示维数的名字、Display Names and Values 表示显示维数的名字和值、Display Name, Values, and Totals 表示显示维数的名字、值和汇总数、Run Time

Display Only 表示只有在运行时而不是设计时才显示维数的相关内容。

该标签页的显示如图 10-2 所示。

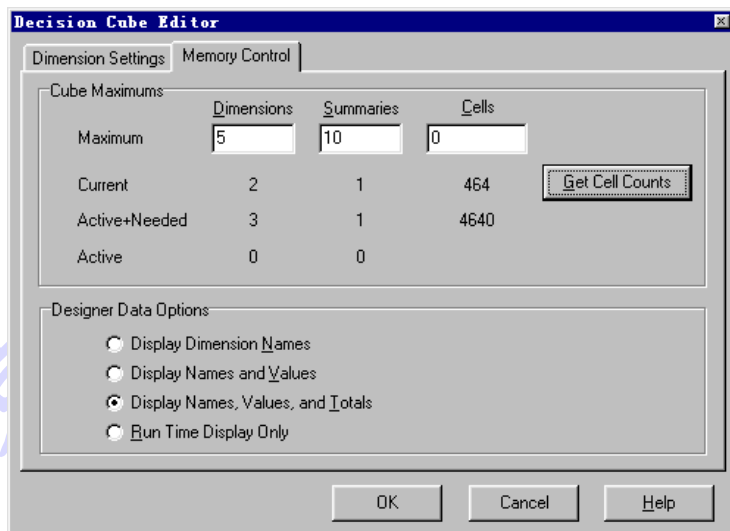


图 10-2 Memory Control 标签页

- **MaxCells** 设置 TDecisionCube 组件显示的单元格的最大数量，与图 10-2 中 Cells 编辑框的作用相同，二者相互影响。
- **MaxDimensions** 设置 TDecisionCube 组件显示的最大维数，与图 10-2 中 Dimensions 编辑框的作用相同，二者相互影响。
- **MaxSummaries** 设置 TDecisionCube 组件显示的汇总字段的最大数量，与图 10-2 中 Summaries 编辑框的作用相同，二者相互影响。
- **Name** 设置 TDecisionCube 组件的名字。
- **ShowProgressDialog** 如果该选项设置为 True，则在激活 TDecisionCube 组件对应的数据集时，会显示一个进度条指示当前进行的计算进度。
- **Capacity** 指定表格数据可以使用的内存的最大数量。
- **DimensionCount** 该属性可以在运行时指定表格的维数。
- **SummaryCount** 该属性可以在运行时指定表格中汇总字段的数量。

## 2. TDecisionCube 组件主要的方法

- **GetDetailSQL** 该方法返回的一个 SQL 语句可以产生一个数据集，用于表示 TDecisionCube 组件使用的部分数据的记录。该方法的语法如下：  

```
function GetDetailSQL( ValueArray: TSmallIntArray; SelectList: String;
bActive: Boolean): string;
```
- **GetSQL** 返回一个 SQL 语句，可以用来查看 TDecisionCube 组件使用部分数据。该方法的语法如下：  

```
function GetSQL( ValueArray: TSmallIntArray; bActive: Boolean): String;
```
- **ShowCubeDialog** 用于显示 TDecisionCube 组件的编辑器，可以重新设置该组件使用的维数及汇总字段等。与 DimensionMap 属性打开的编辑器相同，只是该属性在设计阶段对该面板设置，而 ShowCubeDialog 方法是在运行阶段对其进行设置。

- **GetDimensionName** 用于获得维数的名字，其语法如下：  

```
function GetDimensionName(Dimension: Integer): String; virtual;
```
  - **GetMemoryUsage** 获得 TDecisionCube 组件对应表格中的数据已经使用的内存数量。语法如下：  

```
function GetMemoryUsage: Integer;
```
  - **GetSummaryName** 返回活动的汇总字段的名字，语法如下：  

```
function GetSummaryName(ISum: Integer): String; virtual;
```
3. TDecisionCube 组件主要的事件
- **AfterClose** 当表格中的数据不再显示之后触发该事件。
  - **AfterOpen** 当表格中的数据被激活以后触发该事件。
  - **BeforeClose** 当表格中的数据不再显示之前触发该事件。
  - **BeforeOpen** 当表格中的数据被激活以前触发该事件。
  - **OnLowCapacity** 如果表格数据使用的内存超过 Capacity 属性设置的内存时触发该事件。
  - **OnRefresh** 当表格的维数改变时立即触发该事件。

#### 10.1.2 TDecisionQuery 组件

该组件是专门用于 Decision Cube 面板的组件，为这些组件提供一个数据集，该数据集通过在 SQL 属性中设置 SQL 语句来实现数据的查询，其中的 SQL 必须包含使用 SUM 进行数据汇总的字段，同时还必须使用 Group By 来实现数据的分组统计。TDecisionQuery 组件提供的数据主要传递到 TDecisionCube，然后再通过该组件传递到其他的组件。

当然，也可以使用 TTable、TQuery 等组件作为 TDecisionCube 组件的数据集，但是这些组件没有 TDecisionQuery 组件那么容易使用，所以一般不要使用这些组件。

##### 1. TDecisionQuery 组件主要的属性

- **Active** 决定是否激活该数据集。
- **AutoCalcFields** 决定何时触发 OnCalcFields 事件及合适计算查找字段的值。
- **AutoRefresh** 决定当数据库中的数据改变时，是否自动更新表格中显示的数据。
- **CachedUpdates** 指定是否激活数据集的缓冲区更新。
- **Constrained** 指定当通过 SELECT 建立了一个结果集时，是否允许执行不符合该结果集的更新及插入操作，该属性只适用于 Paradox 和 dBASE 基表。
- **Constraints** 指定编辑数据时必须遵守的记录级的限制。
- **DatabaseName** 指定使用的数据库的名字或别名。
- **DataSource** 指定另一个数据源（不是当前数据集使用的数据源），当前字段的数据可以从该数据源中提取。
- **Filter** 设置筛选条件，一般在 SQL 属性中设置 Where 代替该属性。
- **Filtered** 如果该属性为 True，则表示可以使用 Filter 属性中设置的筛选条件。
- **ParamCheck** 如果在运行时改变 SQL 属性时，该属性决定是否应重新产生一个查询的参数列表。
- **Params** 设置 SQL 属性中包含的参数。



- RequestLive 只有将该属性设置为 True, 才可以修改数据集的数据; 否则数据是只读的。
  - SQL 设置一个 SELECT 语句来建立一个数据集。注意, 该 SQL 语句中必须有一个数据字段使用 SUM 函数进行数据汇总, 同时还要使用 Group By 子句来实现数据的分组。
  - UniDirectional 决定数据集的数据是否是单方向的。
  - UpdateMode 该属性设置当更新一个 SQL 数据库时, BDE 符合查找记录。
  - UpdateObject 指定一个 TUpdateSQL 更新组件来更新一个只读的结果集, 此时必须将 CachedUpdates 属性设置为 True。
2. TDecisionQuery 组件主要的方法
- ExecSQL 该方法用于运行一个 SQL 语句, 这些 SQL 语句一般不返回结果集, 比如 INSERT、UPDATE、DELETE 和 CREATE TABLE 等语句。如果 SQL 语句是 SELECT 语句, 一般使用 Open 方法来代替 ExecSQL 方法。
  - GetDetailLinkFields 在列表中显示主从表的关联字段。
  - ParamByName 为某个参数设置参数值, 其语法如下:  

```
function ParamByName(const Value: String): TParam;
```
  - Prepare 使 BDE 和远程的数据库服务器为当前的查询分配资源, 并且执行一些优化。该方法一般用在 ExecSQL 方法前面。
  - UnPrepare 释放使用 Prepare 方法为当前查询分配的资源。
  - Open 激活数据集中的 SQL 语句, 一般只用于 SELECT 语句。
  - Free 销毁当前对象并释放其使用的资源。
3. TDecisionQuery 组件主要的事件
- Before 开头的事件 表示在某种情形出现以前或调用某个方法以前触发该事件。
  - After 开头的事件 表示在某种情形出现以后或调用某个方法以后触发该事件。
  - On 开头的事件 表示在某种情形出现或调用某个方法时触发该事件。

### 10.1.3 TDecisionSource 组件

TDecisionSource 与 TDataSource 组件的作用类似, 只是它是专门用于决策表格和决策图形。它起到将 TDecisionCube 与 TDecisionPivot、TDecisionGrid 和 TDecisionGraph 进行连接的作用。

#### 1. TDecisionSource 组件主要的属性

- ControlType 指定 TDecisionSource 组件如何响应 TDecisionPivot 组件提供的消息, 即如何结合行或列的维数并同时打开。三个属性值的意义见表 10-1。

表 10-1 ControlType 属性的可设置值及意义

属性值	说明
xtCheck	可以改变打开的维数
xtRadio	此时单行或单列的维数都是 1
xtRadionEx	总是打开一维的行和列

- DecisionCube 指定使用的 TDecisionCube 组件。
  - Name 为 TDecisionSource 组件指定一个名字。
  - SparseCols 如果该属性为 False, 则只显示在一行或多行对应的单元格中有汇总数据的列, 而不显示没有汇总数据的列。
  - SparseRows 与 SparseCols 属性作用相似, 不过该属性为 False 时不显示任何一行对应的单元格都没有汇总数据的空行。
  - CurrentSum 指示当前汇总的索引号。
2. TDecisionSource 组件主要的方法
- CloseDimIndexRight 用于关闭一个维的显示。
  - DrillDimIndex 将所有值汇总到一起。
  - GetDataAsString 将一个单元格的数据以字符串的形式显示。
  - GetDataAsVariant 获得单元格中数据的值。
3. TDecisionSource 组件主要的事件
- OnAfterPivot 当表格的轴心改变后触发该事件。
  - OnBeforePivot 当表格的轴心改变前触发该事件。
  - OnLayoutChange 当激活的维数改变时触发该事件。
  - OnNewDimensions 当 TDecisionCube 组件提供的数据改变时触发该事件。
  - OnStateChange 当 TDecisionCube 组件属性改变时触发该事件。
  - OnSummaryChange 当 CurrentSum 属性值改变时触发该事件。

#### 10.1.4 TDecisionPivot 组件

TDecisionPivot 与 TDBNavigator 组件的作用相似, 用于表格中显示数据的导航, 并控制 TDecisionGraph 和 TDecisionGrid 组件的显示状态, 它可以用来合并或展开某一维。

1. TDecisionPivot 组件主要的属性
- Align 决定该组件如何在其父控件中调整位置。
  - BevelInner 决定 TDecisionPivot 组件内边缘是否需要凹凸。
  - BevelOuter 决定 TDecisionPivot 组件的外边缘是否需要凹凸。
  - ButtonHeight 设置维数按钮的高度。
  - ButtonWidth 设置维数按钮的宽度。
  - Color 指定该控件的背景颜色。
  - DecisionSource 指定一个 TDecisionSource 组件作为数据的源对象。
  - Enabled 用于控制该控件是否响应鼠标、键盘和定时器的的事件。
  - GroupLayout 指定按钮排列的形式, xtHorizontal 表示水平排列、xtVertical 表示垂直排列、xtLeftTop 表示自左上角开始排列。
  - Hint 设置组件的提示信息。
  - ShowHint 用于决定是否显示 Hint 属性中设置的提示信息。
  - Visible 决定该组件是否是可见的。
2. TDecisionPivot 组件主要的方法
- SetBounds 重新设置按钮的大小。其语法如下:

```
procedure SetBounds(Left, Top, Height, Width: Integer);
override;
```

- GetControlsAlignment 返回 Alignment 属性的值。
  - Hide 隐藏该控件。
  - Show 显示该控件。
3. TDecisionPivot 组件主要的事件
- OnClick 单击该控件时触发该事件。
  - OnDblClick: 在该控件上面双击鼠标左键时触发该事件。
  - OnDragDrop 当拖拉该控件并释放后触发该事件。
  - OnDragOver 当用户在该控件上面拖拉鼠标时触发该事件。
  - OnEndDrag 当用户在该控件终止拖拉鼠标时触发该事件。
  - OnEnter 当该控件被激活时触发该事件。
  - OnExit 当该控件从激活状态进入非激活状态时触发该事件。
  - OnResize 当改变该控件的大小后触发该事件。
  - OnStartDrag 当开始在该控件上拖拉鼠标时触发该事件。

#### 10.1.5 TDecisionGrid 组件

TDecisionGrid 类似于 TDBGrid 组件，以表格的形式显示数据库的数据，但是，该表格可以使用多维形式来显示数据库的数据，同时还可以任意改变维数来显示不同的统计信息。

##### 1. TDecisionGrid 组件主要的属性

- CaptionColor 指定包含行和列的维数所在单元格的背景颜色。
- DataColor 指定包含汇总数据的单元格的背景颜色。
- DataSumColor 指定表示所有维数部分和的单元格的背景颜色。
- DecisionSource 设置使用的 TDecisionSource 组件。
- GridLineColor 设置表格边线的颜色。
- GridLineWidth 设置表格边线的宽度。
- Hint 设置帮助信息。
- ShowHint 该属性设置为 True，才能显示 Hint 中提供的帮助。
- Visible 决定是否显示该组件。

##### 2. TDecisionGrid 组件主要的方法

- CellDrawState 从表格的数据单元格中提取数据。
- CellRect 获得表格中一个单元格的屏幕坐标。
- CellValueArray 返回所有具有汇总数据的单元格对应维数字段的值。

##### 3. TDecisionGrid 组件主要的事件

- On 开头的事件 TDecisionGrid 组件的所有事件都以 On 开头，表示在某种情形出现时或调用某种方法时触发该事件。

#### 10.1.6 TDecisionGraph 组件

TDecisionGraph 类似于 TChart 组件，可以使用图形来显示表格化的数据，该图形将第

一行的维数作为 X 轴，将第一列的维数作为 Y 轴。在应用程序运行时，该图形会根据统计数据的变化而改变。

#### 1. TDecisionGraph 组件主要的属性

- AxisVisible 该属性设置为 True，则会显示 TDecisionGraph 组件的坐标轴。
- BackColor 设置 TDecisionGraph 组件背景的颜色，默认设置值 clDefault 为灰色。
- BackImage 设置 TDecisionGraph 组件的背景图像。
- BottomAxis 对应 Chart 属性面板的 Axis 标签页中 Axis 部分的 Bottom 选项，Chart 属性设置面板见图 10-8。
- BottomWall 对应 Chart 属性面板的 Walls 标签页的 Bottom Wall 页面。
- DecisionSource 设置该组件使用的 TDecisionSource 组件。
- Foot 对应 Chart 属性面板的 Titles 标签页。
- Frame 设置 TDecisionGraph 组件框架的属性，其设置面板如图 10-3 所示。该面板包括如下几部分：
  - ◇ Visible 决定是否显示框架。
  - ◇ Width 设置框架的宽度。
  - ◇ Style 设置框架的风格，有如下选项：
    - ☆ Solid 框架使用实线绘制。
    - ☆ Dash 框架使用虚线绘制。
    - ☆ Dot 框架使用小点绘制。
    - ☆ Dash Dot 框架使用线段、小点间隔绘制。
    - ☆ Dash Dot Dot 框架使用线段、小点、小点间隔绘制。



图 10-3 设置框架的属性

- ◇ Color 设置框架的颜色。
  - Gradient 对应 Chart 属性面板的 Panel 标签页。
  - LeftAxis 对应 Chart 属性面板的 Axis 标签页中 Axis 部分的 Left 选项。
  - LeftWall 对应 Chart 属性面板的 Walls 标签页的 Left Wall 页面。
  - Legend 对应 Chart 属性面板的 Legend 标签页。
  - Name 设置 TDecisionGraph 组件的名字。
  - RightAxis 对应 Chart 属性面板的 Axis 标签页中 Axis 部分的 Right 选项。
  - SeriesList 对应 Chart 属性面板的 Series 标签页。
  - Title 对应 Chart 属性面板的 Title 标签页。
  - TopAxis 对应 Chart 属性面板的 Axis 标签页中 Axis 部分的 Top 选项。

#### 2. TDecisionGraph 组件主要的方法

- GetASeries 返回 Chart 中第一个活动的序列，如果 Chart 没有序列，或没有活动的序列，则该方法返回 NIL。
- GetAxisSeries 该函数返回依赖于特定坐标轴的第一个序列，如果某个坐标轴没有序列，则返回值为 NIL。

- Hide 隐藏该控件。
  - Show 显示该控件。
  - SaveToBitmapFile 将当前的 Chart 保存为一个 BMP 文件。
  - SaveToMetafile 将当前的 Chart 保存为一个 WMF 文件
3. TDecisionGraph 组件主要的事件
- On 开头的事件 在出现某种情形或调用某种方法时触发一个事件。

## 10.2 Decision Cube 应用实例

下面通过一个实例说明 Decision Cube 面板组件的用法，该实例将使用 Decision Cube 面板中所有的组件。

### 10.2.1 实现步骤

该实例的实现步骤如下：

1. 使用 File|New|Application 菜单新建一个应用程序。
2. 在表单 Form1 中添加组件。

打开 Decision Cube 面板，分别选择该面板的 6 个组件，然后将其添加到表单中，名字分别是 DecisionQuery1、DecisionCube1、DecisionSource1、DecisionGrid1、DecisionGraph1、DecisionPivot1。其中，DecisionQuery1、DecisionCube1、DecisionSource1 等 3 个组件放置在表单的顶部，DecisionGrid1、DecisionGraph1 组件放置在表单的中间，DecisionPivot1 组件放置在表单的底部，同时通过拖拉适当调整表单、DecisionGrid1、DecisionGraph1、DecisionPivot1 等组件的位置及大小。

3. 设置组件的属性。

各个组件属性的设置值见表 10-2。

表 10-2 设置各个组件的属性

组件	属性	设置值	说明
DecisionQuery1	Active	True	
	DatabaseName	DBDEMOS	
	SQL	Select EventNo,CustNo,NumTickets,Sum(Amt_Paid) from reservat group by EventNo,CustNo,NumTickets	该 SQL 语句形成的表格是三维的，这三维分别是 EventNo,CustNo,NumTickets，Amt_Paid 字段则用来进行合计
DecisionCube1	DataSet	DecisionQuery1	设置数据集
DecisionSource1	DecisionCube	DecisionCube1	设置使用的 TDecisionCube 组件
DecisionGrid1	DecisionSource	DecisionSource1	设置数据源
DecisionGraph1	DecisionSource	DecisionSource1	
DecisionPivot1	DecisionSource	DecisionSource1	

从以上属性的设置可以看出，Decision Cube 面板的组件显示数据的方法与通过 BDE、ADO 等面板的组件开发数据库的方法相似，只是现在 TDecisionSource 没有直接调用

TDecisionQuery 组件，而是将 TDecisionCube 组件作为中间件，该实例的数据流程如下：

数据库数据 → DecisionQuery1 → DecisionCube1 → DecisionSource1 → Decision-Grid1  
(DecisionGraph1 或 DecisionPivot1) → 显示或控制数据

以上所有组件的属性设置完毕后，只按下 DecisionPivot1 的 CustNo 按钮时，界面的显示如图 10-4 所示，从图中可以看出数据的统计值与右边的图例、坐标值的对应关系。

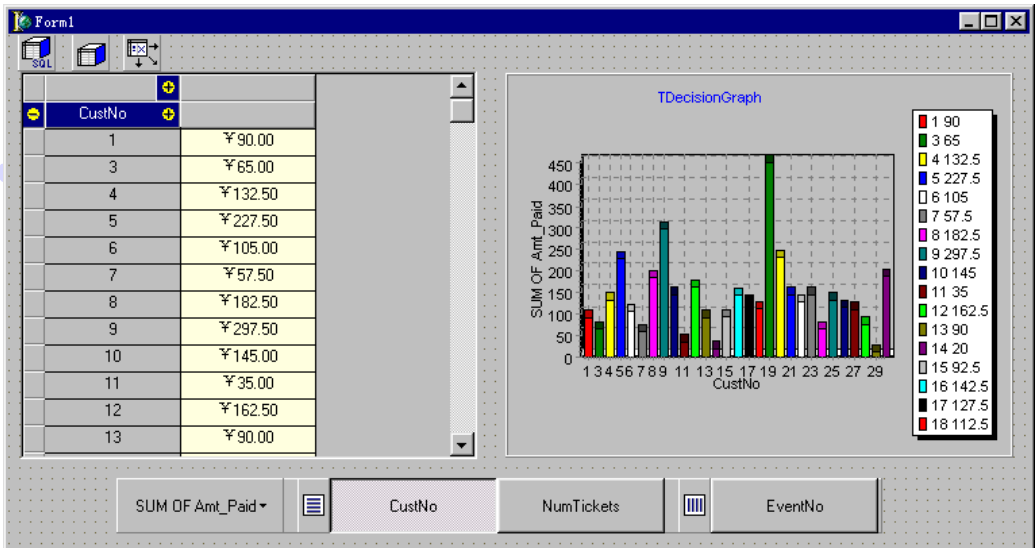


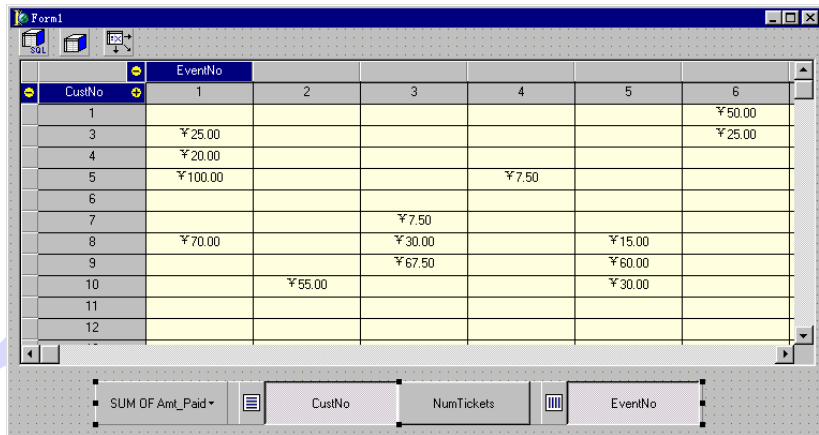
图 10-4 只按下 CustNo 按钮时显示的界面

从图 10-4 中可以看出，TDecisionPivot 组件会根据 TDecisionQuery 的 SQL 属性设置的 SQL 语句中查询的字段数量来决定按钮的个数，每个按钮都对应表格或图形中的一维或汇总的数据。比如图 10-4 中，SUM OF Amt\_Paid 按钮用于 Amt\_Paid 字段的合计，该合计数据会根据当前显示的维数而变化；CustNo 和 NumTickets 两个按钮对应 CustNo 和 NumTickets 两个字段，它们对应的数据按行进行显示，CustNo 左边的小图标说明了这一点；EventNo 按钮对应 EventNo 字段，它对应的数据按列进行显示。CustNo、NumTickets、EventNo 等按钮中任何一个或多个被按下，则会显示出其对应的字段，并且作为表格或图形的维数。比如，现在隐藏起 DecisionGraph1，将 DecisionGrid1 的宽度加大，此时按下 CustNo 和 EventNo 两个按钮，则表格会将 CustNo 和 EventNo 两个字段作为纵向和横向坐标，并显示对应这两个坐标的 Amt\_Paid 字段的合计信息，如图 10-5 所示。

其实，如果不使用 TDecisionPivot 组件的各个按钮，单击表格中各个维数右边黄色的加号“+”和减号“-”按钮，也同样可以关闭或打开表格的某一维。

如果修改某一维对应的统计图形使用的模板，则可以改变该维对应的图形的风格，这些风格可以通过 Chart 属性设置面板的 Series 标签页进行设置，比如 EventNo 对应的图形风格设置成粗线条，则只选择 EventNo 按钮时界面的显示如图 10-6 所示。





	EventNo					
CustNo	1	2	3	4	5	6
1						¥50.00
3	¥25.00					¥25.00
4	¥20.00					
5	¥100.00			¥7.50		
6						
7			¥7.50			
8	¥70.00		¥30.00		¥15.00	
9			¥67.50		¥60.00	
10		¥55.00			¥30.00	
11						
12						

图 10-5 按下 CustNo 和 EventNo 两个按钮时显示的合计信息

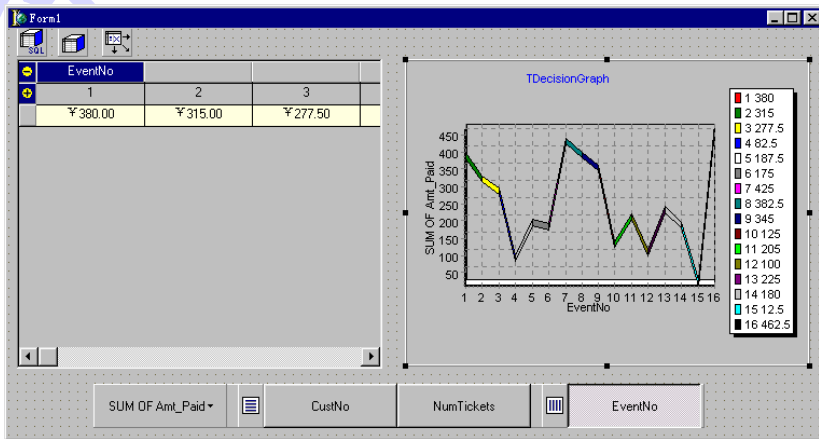


图 10-6 只选择 EventNo 按钮时界面的显示

如果使用细线来绘制 NumTickets 对应的图形，则运行程序后，DecisionPivot1 组件右边的三个按钮同时按下时，界面的显示如图 10-7 所示。

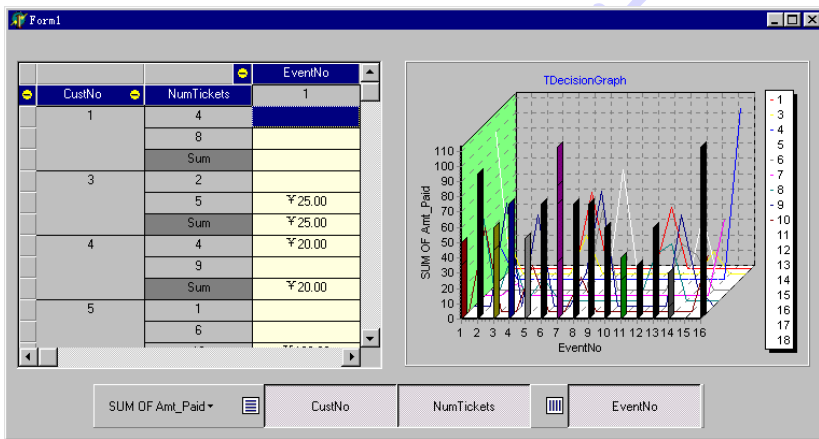


图 10-7 同时按下 DecisionPivot1 组件右边的三个按钮时显示的界面

10.2.2 Chart 属性设置面板

设计阶段在 DecisionGraph1 组件上单击鼠标右键，会弹出一个上下文菜单，从中选择 Edit Chart 菜单，此时会打开 Chart 属性设置面板，如图 10-8 所示。

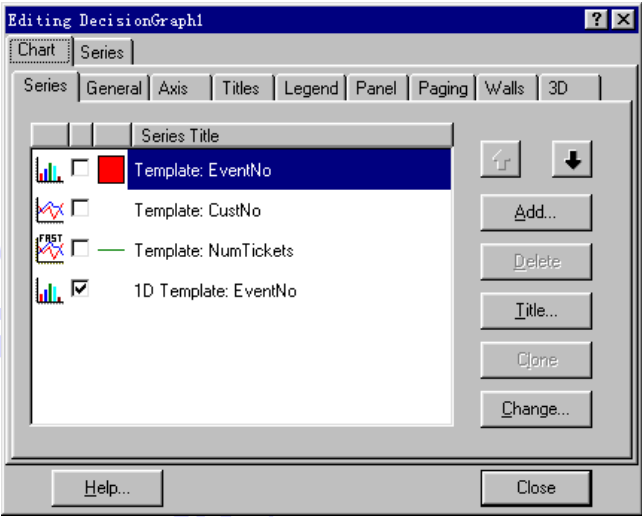


图 10-8 Chart 属性设置面板

从图中可以看出，该属性面板与 TChart、TDBChart 组件的 Chart 属性面板完全一样，设置方法也一样。该面板实际上包括 Chart 和 Series 两个标签页，而它们又分别包括多个标签页。

在说明 Chart 属性设置面板各个选项的作用以前，应该先搞清楚通过该属性面板建立的最最终显示图形中各部分的作用。统计图形中各部分的作用如图 10-9 所示。

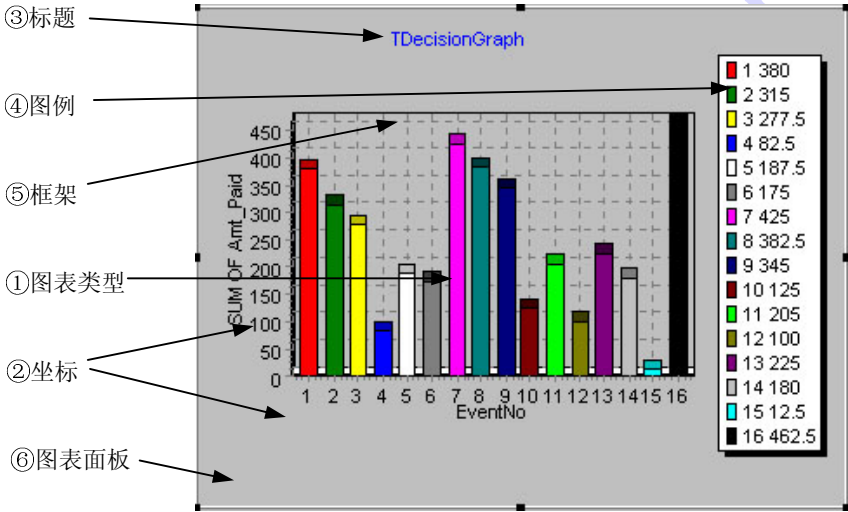


图 10-9 统计图形的显示

下面分别说明 Chart 属性设置面板各部分的作用。

1. Chart 标签页 该标签页用于设置图表的各种属性，它包括 9 个标签页，这些标签页的作用如下：

- Series 设置图表各个坐标序列对应的类型，可以是直方图、饼图或折线图等。
- General 设置图表的基本属性，比如打印预览、以图形文件进行输出、距离边界宽度、是否可以放大、是否添加滚动条等。
- Axis 设置坐标轴的一些属性。
- Titles 设置标题的一些属性。
- Legend 设置图例的一些属性。
- Panel 设置图表所在面板的一些属性。
- Paging 设置图表各个页面的属性。
- Walls 设置左边、下面和后面使用的墙的属性。
- 3D 设置图表的 3 维显示方式，比如可以对图表进行旋转、偏移或透视等。

(1) Series 标签页 设置使用的图表类型。每种不同的字段可以使用不同的图表类型，默认情况下都使用直方图。要改变一个字段的图表类型很简单，即在 Series 标签页的图表列表中选择要修改的字段，然后单击右边的 Change 按钮，此时会打开图表类型选择面板，如图 10-10 所示，可以从中选择粗折线图（Line）、细折线图（Fast Line）、直方图（Bar）、饼图（Pie）及其他类型的图表。然后单击 OK 按钮确认，即可修改图表的类型。

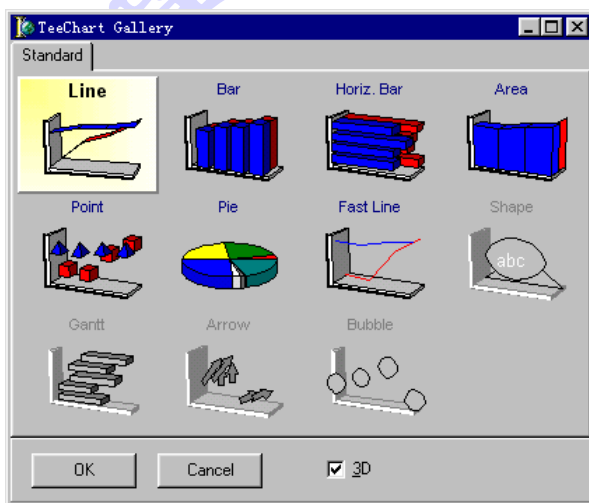


图 10-10 选择图表的类型

其他按钮作用如下：

- Add 按钮 单击该按钮，可以为当前选择的字段新增加一个图表类型，即使用两种以上的方式来显示一个字段的的数据。
- Delete 按钮 删除列表中当前选择的图表。
- Title 设置一个图表的标题。
- Clone “克隆”一个新添加的图表。
- 上、下箭头按钮 改变图表显示的顺序。

(2) General 该标签页的显示如图 10-11 所示。

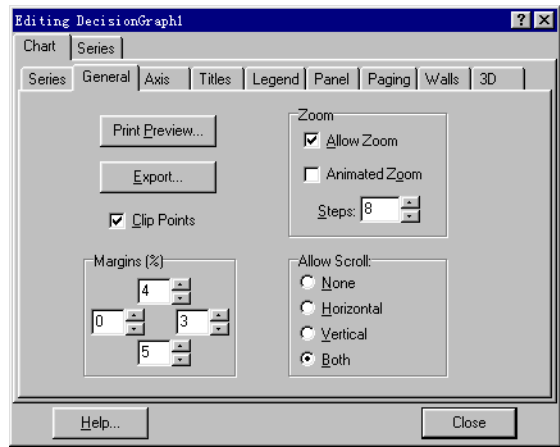


图 10-11 General 标签页

该标签页各个选项的作用如下：

- **Print Preview** 按钮 用于预览图表的打印效果。
- **Export** 按钮 该按钮用于将当前的图表输出到一个文件或拷贝到剪贴板上，打开的对话框如图 10-12 所示。Format 一栏用于选择保存图形的类型。

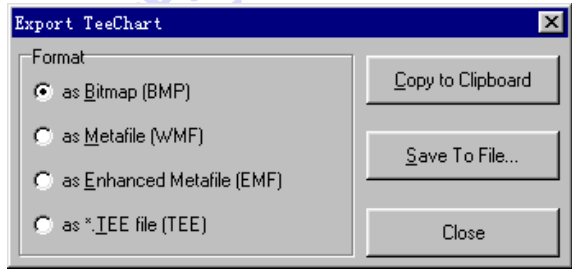


图 10-12 选择输出图形的方式及格式

- **Margins** 设置图表离上、下、左、右边界的距离，这些值设置得越大则图表越小。
- **Zoom** 该部分选项用于放大图表的显示。
- **Allow Scroll** 决定是否显示水平及垂直滚动条。

(3) **Axis** 该标签页的显示如图 10-13 所示。该标签页实际上分为两部分：左边的 Axis 部分的 5 个单选按钮 (Left、Right、Top、Bottom、Depth) 确定当前设置属性的坐标轴，右边的 6 个标签页 ( Scales、Title、Labels、Ticks、Minor、Position) 则用于设置当前选择的坐标轴的属性。

另外，**Show Axis** 决定所有的坐标轴是否可见；**Visible** 则设置当前选择的坐标轴是否可见，如果没有选中 **Show Axis** 选项，即使某个坐标轴选中了 **Visible** 选项，也不会显示该坐标轴。

右边几个标签页的作用如下（以下面板均以左边的坐标轴为例）：

**Scales** 设置当前选择坐标轴的伸缩度，选择 **Automatic** 后会自动根据显示图形的范围伸缩。如果不选择 **Automatic** 选项，则可以手动设置坐标轴的最大值和最小值。如果选择 **Logarithmic** 选项，则表示按照对数值计算坐标；如果选择 **Inverted**，则所有坐标的值会反转，此时会反转图形。该标签页的显示如图 10-14 所示。

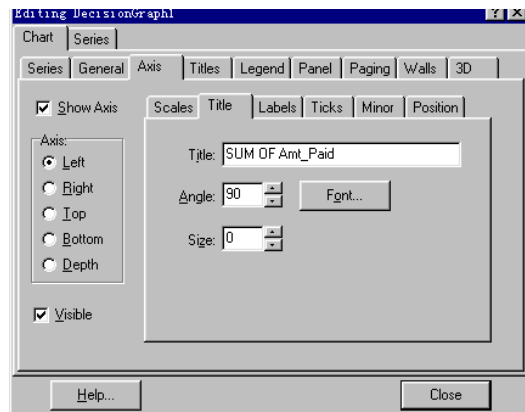


图 10-13 Axis 标签页

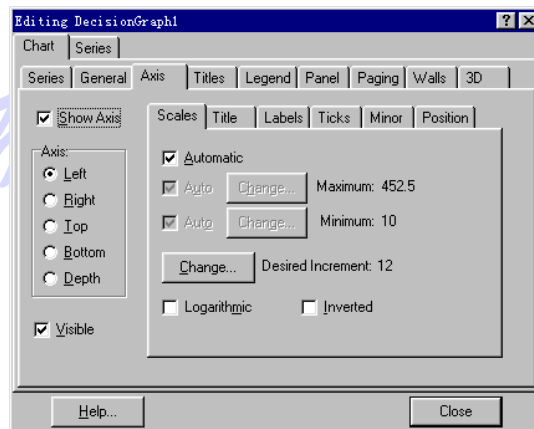


图 10-14 设置坐标伸缩度的标签页

- **Title** 坐标轴的该标签页用于设置当前选择坐标轴的标题名称、标题的偏转角度、使用的字体及标题的大小，Size 中设置的值越大，则标题使用的空间就越大，显示图形的区域就越小。前面实例中左边的坐标轴对应的标题设置面板如图 10-15 所示。

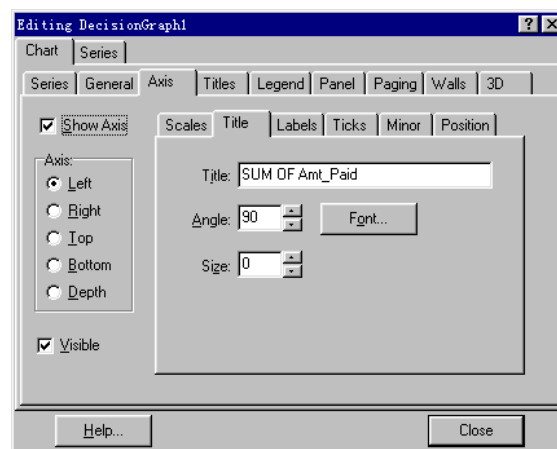


图 10-15 设置坐标轴的标题属性

- **Labels** 设置坐标值的一些属性，比如 **Visible** 用于显示坐标值、**Round First** 表示对第一个小数进行四舍五入、**Min.Separation** 用于设置坐标值的间隔、**Angle** 及 **Size** 用于设置坐标值的偏转角度及占用空间的大小、**Style** 用于设置坐标值的显示风格等。该标签页的显示如图 10-16 所示。

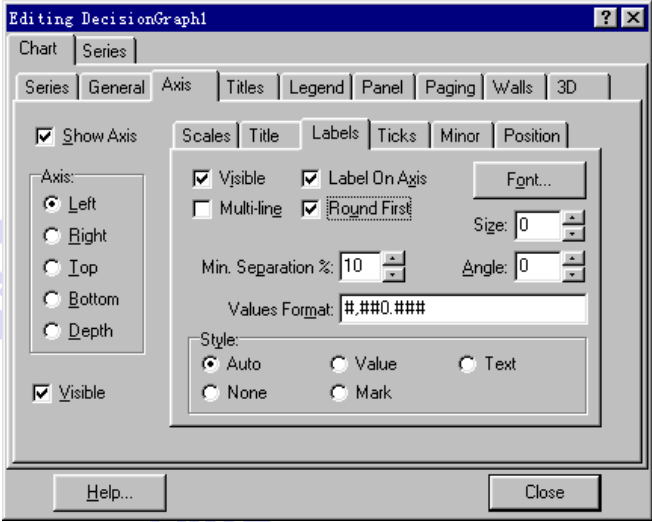


图 10-16 设置坐标值的一些属性

- **Ticks** 标签页 用于设置坐标轴边界、图形中的网格线边界、坐标值到坐标轴间的指示线的长度等属性。该标签页的显示如图 10-17 所示。

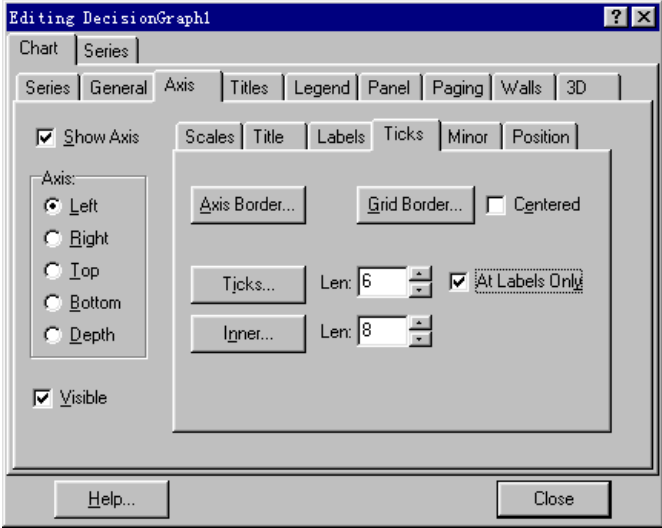


图 10-17 设置坐标轴上坐标值指示线的间隔

- **Minor** 用于设置坐标轴上坐标线的最小间隔，可以设置这些间隔线的长度、数量等。该标签页的显示如图 10-18 所示。



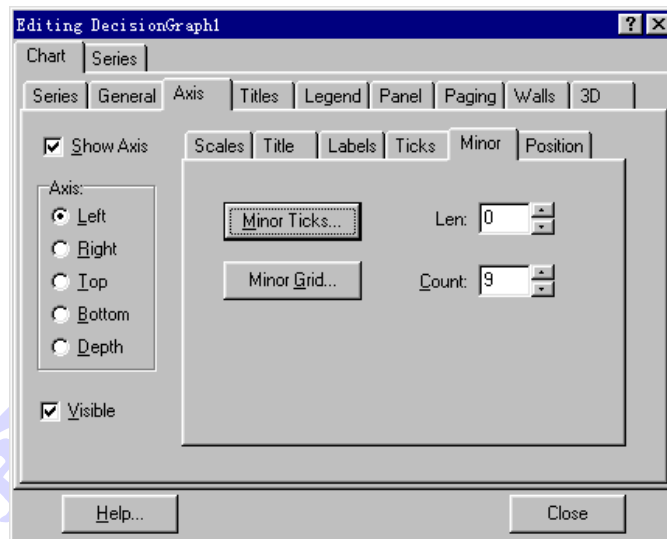


图 10-18 设置坐标轴上的间隔线

- **Position** 设置坐标轴的位置，同时也设置图形的位置。Position 选项设置坐标轴的起始位置；Start 设置当前坐标的起始位置，一般设置为 0，最大值不能超过 100；End 设置坐标的终止位置，一般为 100，最小值不能小于 0。该标签页的显示如图 10-19 所示。

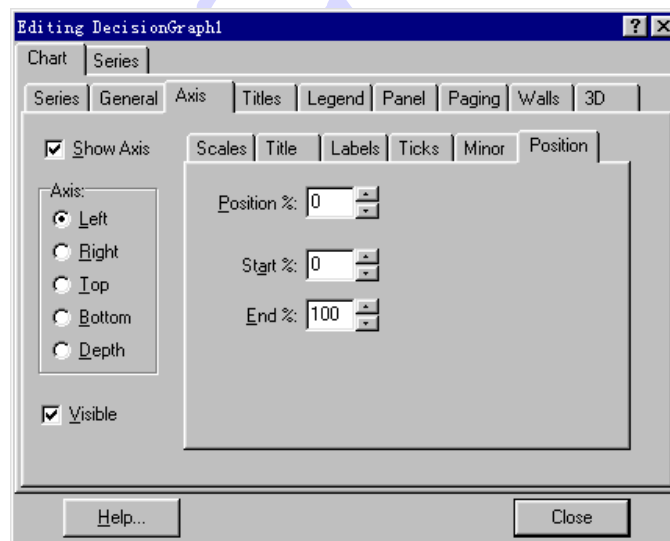


图 10-19 设置坐标的位置

(4) **Titles** 设置图表标题的一些属性。比如使用 **Visible** 选项控制其是否可见、使用 **Alignment** 属性控制其位置、**Back Color** 控制其背景颜色、**Font** 控制标题字体等。该标签页的显示如图 10-20 所示。



图 10-20 设置图表的标题属性

(5) Legend 设置图例的一些属性。每个图表中会默认一个图例，图例中一般会显示某个坐标轴不同值对应的颜色、坐标值、实际值等内容，但是也可以改变其风格。Legend 标签页就可以改变图例的一些属性，比如 Legend Style 选项设置图例的类型、Text Style 设置图例中文本内容的风格、图例的位置、使用的框架、字体、阴影等一些属性。该面板的显示如图 10-21 所示。

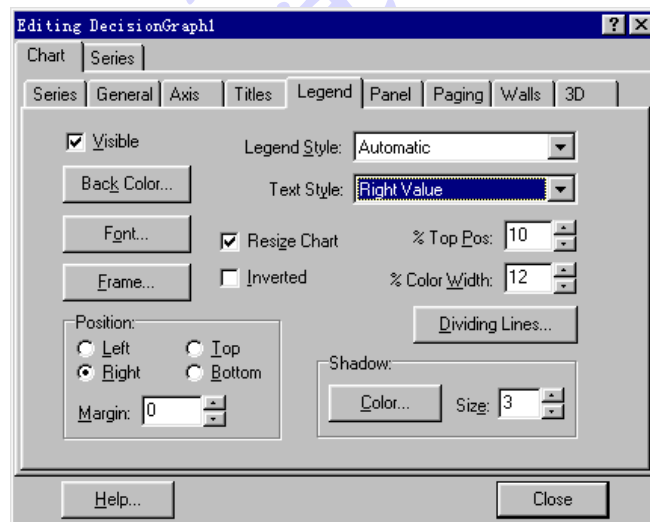


图 10-21 设置图例属性的标签页

(6) Panel 设置图表所在面板的一些属性。比如 Bevel Inner 及 Bevel Outer 选项设置面板凹凸、Panel Color 设置面板的颜色、两个 Width 选项分别设置面板距离边界的宽度及边界的宽度、Back Image 可以为图表面板设置一个背景图像、Gradient 则可以将其背景设置为一种渐变色。该标签页的显示如图 10-22 所示，设置了渐变色的面板如图 10-23 所示。

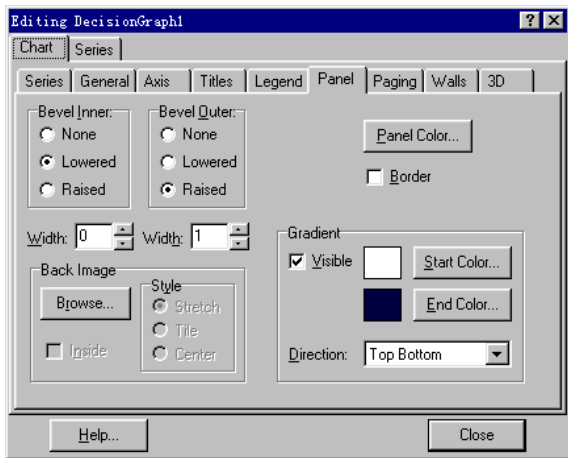


图 10-22 设置图表面板的属性

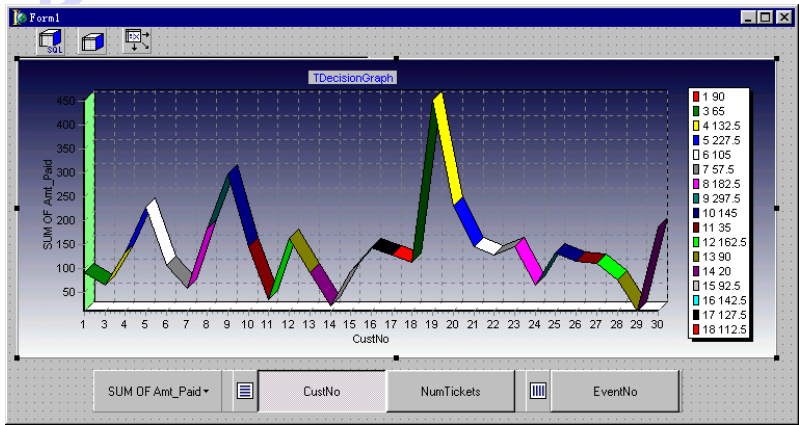


图 10-23 图表面板设置了渐变色后的显示

(7) **Paging** 设置图表各个页面的属性。比如通过改变 Points per Page (单个页面像素点的数量)选项中的值,可以改变图表显示的范围,其中 0 表示使用合适的默认值。Current Page/Total 表示当前页面在总页数中的比例。该面板的显示如图 10-24 所示。

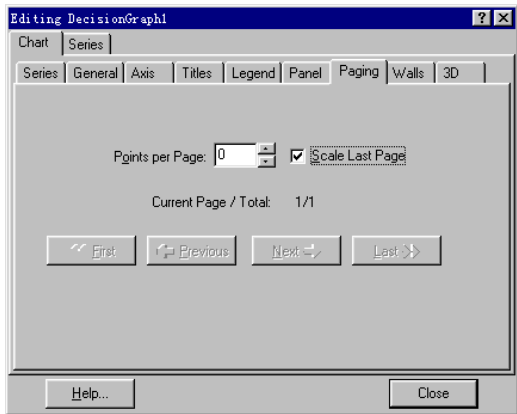


图 10-24 Paging 标签页的显示

(8) Walls 设置左边、下面和后面使用的阴影部分的属性，比如使用的背景颜色、边界类型及样式、阴影是否透明等。通过这些阴影，可以使图表更具有立体感。该标签页的显示如图 10-25 所示。

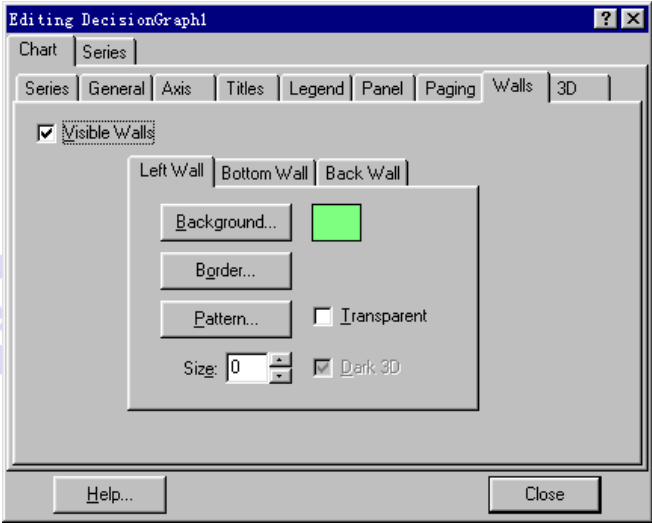


图 10-25 设置图表各种阴影的属性

(9) 3D 设置图表的 3 维显示方式，比如可以对图表进行旋转、偏移或透视等，该面板的显示如图 10-26 所示。通过该标签页的设置，可以从多个角度观察图表。

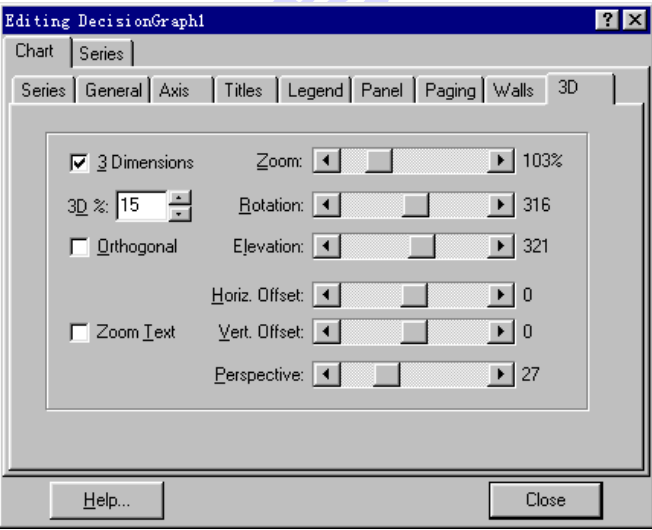


图 10-26 对图表进行 3 维显示的控制

- 3 Dimensions 选中该选项，则图表会以立体的形状显示。
- 3D% 设置图表立体的程度。
- Orthogonal 图表以对数形式显示。
- Zoom Text 放大图表坐标值及标题等文本内容。
- Zoom 放大或缩小整个图表的显示。

- Rotation 旋转整个图表。
- Elevation 为图表设置一定的仰角。
- Horiz Offset 设置图表的水平偏移量。
- Vert Offset 设置图表的垂直偏移量。
- Perspective 设置图表的透视效果。

例如，以上选项按照图 10-26 中显示的数据进行设置后，图表的显示如图 10-27 所示。

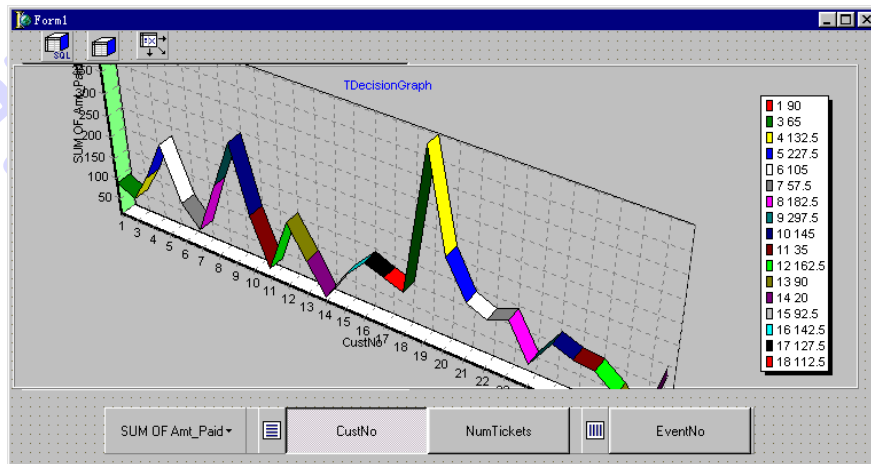


图 10-27 对图表进行 3D 标签页的设置后的显示

注意：图表各个标签页的选项都可以在设置过程中观察对图表的影响。

## 2. Series 标签页

该标签页用于设置每维坐标使用的模板的一些属性，可以从下拉框中选择需要设置的模板。然后通过 Formt、Point、General、Marks 标签页分别设置选择模板的格式、点阵属性、一般属性及一些标识符等属性。一个坐标 Formt 标签页的显示如图 10-28 所示。

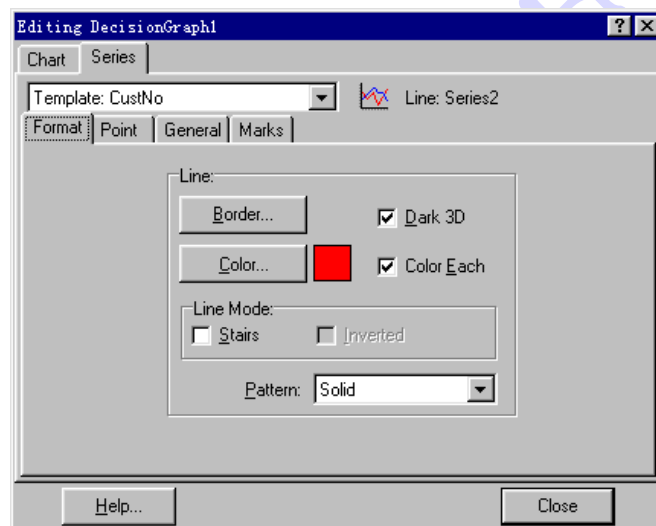


图 10-28 设置坐标模板的一些属性

## 第 11 章 界面设计常用功能

本章将讲述一些开发应用程序时常用的功能，比如如何设计菜单、工具条、状态条等，以及如何使用多媒体及图像、对话框和项目库等。下面分别对这些内容进行说明。

### 11.1 设计菜单

Delphi 中的菜单分为两种：一种是下拉式菜单，该类菜单适合于做系统菜单；另一类是弹出式菜单，这类菜单可以在页面的任何位置弹出，使用起来比较灵活。这两种菜单分别通过 Standard 面板上的 TMainMenu 组件和 TpopupMenu 组件实现。下面通过实例说明如何建立这两种菜单。

#### 11.1.1 下拉式菜单

1. 直接设计下拉式菜单。具体步骤如下：

(1) 新建一个应用程序。

(2) 通过 TMainMenu 组件在应用程序的表单上添加一个下拉菜单组件 MainMenu1。

(3) 在表单中该主菜单组件的图标上面双击鼠标；或者打开 MainMenu1 的对象观察器，在 Items 属性右边的按钮上面单击鼠标，则可以打开菜单编辑窗口。

(4) 此时菜单编辑窗口的右上角会显示一个蓝色的虚线框，表示可以在该虚线框位置插入菜单项，最上面的一排用于插入主菜单项。此时打开对象观察器，在 Caption 中输入“文件(&F)”，在 Name 属性中输入 file1。这样第一个菜单项就建立完毕，其中&F 的作用是在菜单项的右边添加一个带下划线的 F，这样可以使用 Alt+F 打开该菜单。此时在该菜单项的下面及右边各出现一个虚线框，下面的虚线框用于插入该菜单项的下拉菜单；右边的虚线框用于插入新的菜单项。

选择下面的虚线框，再打开对象观察器，将 Caption 属性设置为“新建(&N)”，将 Name 属性设置为“New1”，这样就设计完了第一个下拉菜单。此时在其下面又会添加一个虚线框，按照同样的方法可以继续添加其他菜单。

如果要使用一条横线间隔实现不同功能的下拉菜单，可以选择一个下拉菜单，将其 Caption 属性设置中划线“-”，则该菜单将变为一条间隔线。

在第一个菜单项建立完毕后，可以按照相同的方法建立第二个菜单项及其下拉菜单。

设计了两个菜单后，第一菜单项打开后，菜单编辑器的显示如图 11-1 所示。关闭菜单编辑器后，在表单 Form1 的顶部会自动显示新建立的菜单项，实际上是将 Form1 的 Menu 属性设置为该主菜单组件的名字 MainMenu1。

(5) 添加菜单执行代码。在应用程序中，当用户选择了一个菜单时，则会执行该菜单提供的功能，即其对应的代码。用户选择该菜单就相当于在该菜单上单击了鼠标，所以可以通过菜单的 OnClick 事件来添加其代码。



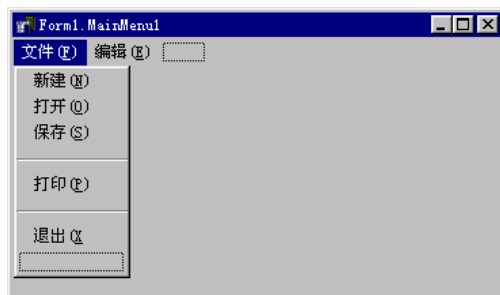


图 11-1 主菜单编辑器

添加该事件对应代码的方法如下：如果打开了菜单编辑器，可以在一个菜单上面单击鼠标，或者打开其对应的对象观察器后，在 Events 标签页 OnClick 事件的右边单击鼠标，可打开代码编辑器；如果在表单中，可以直接选择需要添加代码的菜单，从而打开代码编辑器。

比如，在表单中有一个 TRichEdit 组件 RichEdit1，如果希望选择复制菜单后能够将该编辑框中输入的内容拷贝到剪贴板上，则为“复制”菜单 copy1 添加如下的代码即可：

```
procedure TForm1.copy1Click(Sender: TObject);
begin
    RichEdit1.CopyToClipboard;
end;
```

(6) 菜单的常用属性。在设计菜单时，可能还需要其他功能，比如添加快捷键、合并两个表单的菜单、将菜单设置为单选项等，这些功能都要通过在菜单编辑器中设置某个菜单的属性实现。除了 Caption 和 Name 属性以外，其他几个主要的属性如下：

- Action 如果要通过 ActionList 组件建立菜单，可以将该属性设置为一个 ActionList 组件的名字。
- AutoHotKeys 决定下拉菜单的快捷键是否可以自动设置。
- Bitmap 为菜单设置一个图像，但是不能为顶级菜单项设置图像。
- Break 决定菜单是否显示在新的一列。
- Enabled 决定某个菜单是否被激活。
- GroupIndex 决定如何合并多个表单的菜单。合并的原则是：如果两个窗口中菜单的 GroupIndex 值相同，则新打开窗口的菜单会代替第一个窗口的菜单；如果两个窗口菜单的 GroupIndex 值不同，则两个窗口的菜单都会显示，该属性值大的菜单会显示在后面。
- RadioItem 决定是否将菜单设置成单选项。
- Visible 决定菜单是否可见。

(7) 菜单编辑窗口的弹出菜单。在菜单编辑窗口中单击鼠标右键，会弹出一个上下文菜单，提供了编辑菜单需要的一些功能，这些菜单的作用如下：

- Insert 在当前选择菜单的前面插入一个新菜单。在编辑菜单过程中，会自动在新建立菜单的后面或菜单项的右边插入新的编辑框，用于插入新菜单。
- Delete 删除当前选择的一个菜单。
- Create Submenu 为一个菜单项建立二级子菜单。

- **Select Menu** 该菜单可以打开一个菜单选择窗口，用于选择需要的菜单。
- **Save As Template** 将编辑的菜单保存到一个模板中，以后设计菜单时可以重复使用。
- **Insert From Template** 通过以前建立的菜单模板来设计菜单。
- **Delete Template** 删除以前保存的模板。
- **Insert From Resource** 可以通过 Windows 的一个资源文件建立菜单。

## 2. 通过 TActionList 组件设计菜单。

Delphi 已经定义了一些通用菜单，可以通过 TActionList 组件管理，在设置菜单时可以直接使用这些已经定义好的菜单。其实现步骤如下：

(1) 在表单中添加一个 Standard 面板上的 TActionList 组件，设置名字为 ActionList1。

(2) 在 ActionList1 上面双击鼠标，则会打开命令列表编辑器，然后单击编辑器右上角的图标，从弹出菜单中选择 New Standard Action（在面板上单击鼠标右键也可以选择该菜单），在打开的标准命令列表中选择合适的命令，如果要选择多个命令，可以按下 Shift 或 Ctrl 键。比如选择了 Edit 下的所有命令。单击 OK 按钮，则选择的命令会添加到命令列表编辑器中，如图 11-2 所示。

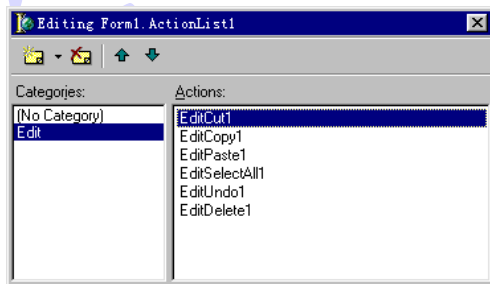


图 11-2 命令列表编辑器

可以从编辑器中选择一条命令，修改其属性。这些命令对应的属性与菜单对应的属性基本相同，所以可按照设置菜单属性的方法设置它们的属性。这些命令的属性设置完毕后，关闭命令列表编辑器。

(3) 重新打开主菜单编辑器，然后新建立一个菜单，在其 Action 属性中选择合适的命令即可。比如要在“编辑”菜单项下添加一个“全选”菜单来选择编辑框中所有内容，则可以在新建立菜单的 Action 属性中选择已经定义的命令 EditSelectAll1，并适当修改 EditSelectAll1 命令的属性，使之符合该菜单的要求，这样该菜单就设计完毕。

如果为某条命令设置了执行代码，则使用该命令的菜单同样会使用这些代码。

### 11.1.2 弹出菜单

设计弹出菜单的方法与主菜单非常相似，具体步骤如下：

1. 在表单中添加一个弹出菜单组件 PopupMenu1。

2. 在弹出菜单 PopupMenu1 上面双击鼠标，则可以打开菜单编辑器，此时会选择第一个菜单编辑框。打开对象观察器的属性面板，在 Caption 属性中输入“复制(&C)”，菜单名字使用默认的 N1，则第一个菜单项建立完毕。

按照同样的方法, 可以将第二个菜单的 Caption 属性分别设置为“剪切(&T)”、“粘贴(&P)”, 这两个菜单使用默认的名字, 这样就建立好了具有三个菜单项的弹出菜单。

3. 添加执行代码。在一个菜单项上双击鼠标, 则会打开代码编辑器, 可以为该菜单添加实现其功能的相应代码。

4. 调用菜单。要使整个表单中的组件都可以使用这些菜单, 则可以在该表单的 PopupMenu 属性中选择 PopupMenu1, 这样运行程序后, 单击鼠标右键, 就可以弹出该菜单。

## 11.2 设计工具条

在应用程序中, 经常在系统菜单项下面建立一个工具条面板, 以按钮图标的形式来实现一些频繁使用的菜单的功能, 通常将这些按钮称为快捷键。

下面以 CLX 应用程序为例, 说明设计工具条及快捷键的步骤如下:

1. 使用 File|New|CLX Application 菜单新建一个跨平台的应用程序。

2. 选择 Common Controls 面板上的 TImageList 组件 (Windows 应用程序使用 Win32 组件面板), 在表单中插入一个图像列表组件 ImageList1, 然后双击该组件, 使用 Add 按钮添加一些工具条按钮可以使用图像或图标。具体添加图像的方法见后续内容。

3. 选择 Common Controls 面板上的 TToolBar 组件, 在表单中插入一个工具条 ToolBar1, 则该工具条会自动显示在表单的顶部, 然后将其 Images 属性设置为 ImageList1。这样在建立新的按钮时, 会自动使用这些图像, 并按顺序分配给各个按钮。

4. 在工具条 ToolBar1 上面单击鼠标右键, 则会弹出一个上下文菜单, 从该菜单中选择 New Button 菜单, 则会在工具条中添加一个新的按钮。

按照同样的方法, 可以添加多个按钮。如果要设置按钮之间的间隔, 则可以选择 New Separator 菜单。

5. 如果已经设置了工具条的 Images 属性, 则建立按钮时会自动为其添加图像作为图标。如果要改变按钮使用图像的的顺序, 可以设置该按钮的 ImageIndex 属性为该图像在 ImageList1 中编号。

6. 设置按钮的对应的事件。快捷按钮一般与某项菜单相对应, 所以应实现该菜单相同的功能。如果按钮设置的是 OnClick 事件, 则只需在该按钮对应对象观察器的 Events 面板上, 在 OnClick 事件中选择某项菜单对应的 OnClick 事件, 即可实现该菜单的功能。比如, 复制菜单对应的 OnClick 事件为 Copy1Click, 则只需在按钮的 OnClick 事件中选择 Copy1Click, 即可实现相同的功能, 也就是说, 单击该快捷按钮与选择对应菜单作用相同。

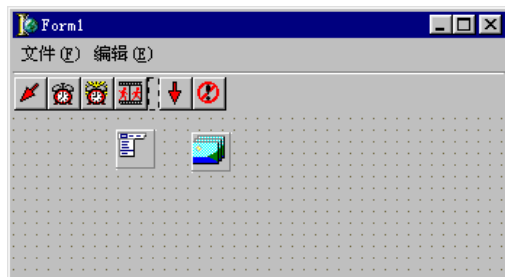


图 11-3 添加工具条后的界面

比如，设计了一个菜单及其对应工具条的界面如图 11-3 所示。

### 11.3 使用状态条

状态条的作用主要是显示应用程序当前执行情况的一些提示信息。通过使用状态条，可以使设计的程序有友好的人机界面，以便于用户操作。

在表单中插入状态条的步骤如下：

1. 选择 Common Controls 面板上的状态条组件 TStatusBar，在表单中插入一个状态条 StatusBar1，该状态条会自动放置在表单的底部。
2. 将 StatusBar1 组件的 AutoHint 属性设置为 True，该属性表示状态条自动显示帮助信息；将 SimplePanel 属性设置为 True，表示只使用具有一行状态条显示状态信息。
3. 在一些组件的 Hint 属性中设置相应的帮助信息，这样程序执行后，当鼠标移动到这些组件上时，就会显示出提示信息。

注意：如果希望组件本身显示帮助信息，则需要将其 ShowHint 属性设置为 True；相反将 ShowHint 属性可以设置为 False，帮助信息就不会显示在组件上面，而是显示在状态条中。

比如，如果在表单的 Hint 属性中输入“这是一个表单。”的帮助信息，运行程序时，如果光标在表单上，就会显示出该帮助信息。如图 11-4 所示。

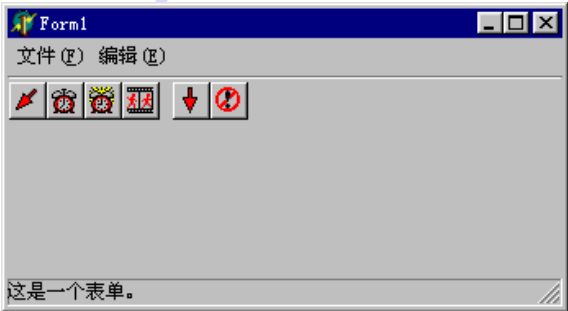


图 11-4 在状态条上显示帮助信息

### 11.4 使用图像及多媒体

图像与多媒体在应用程序开发中占有很大的比重。Delphi 提供了强大的图像显示、图形绘制与多媒体控制等多项功能，可以最大程度地满足开发各种应用程序的要求。

#### 11.4.1 图像与图形

##### 1. 使用 TImageList 组件

TImageList 组件主要用来管理比较小的图标，这些图标可用于工具条的按钮中。使用 TImageList 组件管理图标的方法如下：

- (1) 在表单中添加一个 TImageList 组件 ImageList1。
- (2) 在 ImageList1 上面双击鼠标，则会弹出其管理图标面板，然后单击该面板的 Add 按钮，会弹出文件路径选择窗口，能够选择的文件格式是扩展名为.bmp、.ico、.png、.xpm

的图像或图标文件，比如要使用 Delphi 提供的按钮图标，可以通过上面的路径搜寻按钮找到...\\Borland\\Shard\\Images\\Buttons 目录，此时会列出该目录下的所有图项，然后可以选择合适的图像。

比如要选择前 6 个图像，可以先选择 Abort 文件，然后按下 Shift 键再选择 Arrow1d1 文件，则同时也会选择这两个文件之间的其他 4 个文件，此时文件选择窗口如图 11-5 所示。

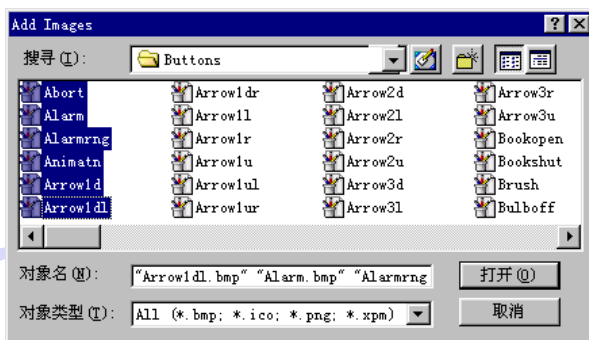


图 11-5 选择添加到 ImageList1 中的图像

(3) 单击图 11-5 中的“打开”按钮，则会显示出一个对话框，如图 11-6 所示，询问是否将较大的图像分裂成两个图像。选择 Yes 则进行分裂，选择 No 则保持原来图像的形狀。此处选择 Yes，因为这些按钮图像由两部分组成，两部分形状一样，只不过一个是彩色的，一个是灰色的。

注意：此时光标的形状是沙漏的形状，直接将其移动到 Yes 按钮上再单击鼠标即可。

下面的图像可能会依次弹出该对话框，分别选择 Yes 即可。选择的所有图像都添加到 ImageList1 的图像列表中时，其显示如图 11-7 所示。

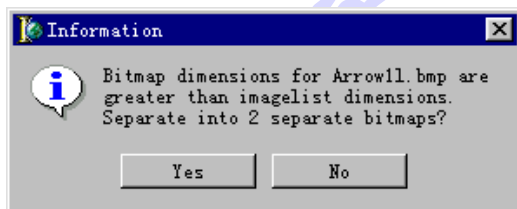


图 11-6 询问是否将较大的图像分裂成两个图像

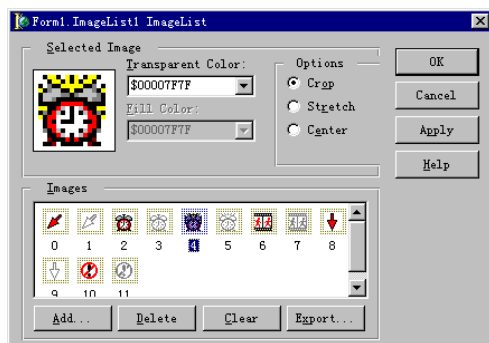


图 11-7 ImageList 图像管理器

(4) 如果希望删除灰色图像，可以先选择一个图像，然后按住 Ctrl 键的同时，再选择

其他的图像，最后单击 **Delete** 按钮，就可以删除这些选择的图像。

从 **ImageList** 图像管理器中可以看出，所有图像都被编了号，第一幅图像的编号是 0。如果删除了一些图像，则剩余的图像会重新编号。前面在讲述工具条时说过，默认情况下，工具条上的按钮使用的图像顺序就是按照这些图像的编号从左向右依次分配给各个按钮。但是也可以通过设置按钮的 **ImageIndex** 属性改变其使用的图像。

(5) **ImageList** 图像管理器中各个选项或按钮的作用。

- **Transparent Color** 设置选择图像中的那种颜色是透明的。
- **Fill Color** 设置选择图像的填充颜色。
- **Crop** 选择该选项，则会裁剪掉图像中过大的部分。
- **Stretch** 如果图像较小，则根据 **ImageList** 规定的大小进行拉伸，此时图像可能会变形。
- **Center** 图像居中显示，不进行拉伸。
- **OK 按钮** 确认当前的设置并关闭窗口。
- **Cancel 按钮** 取消当前的设置。
- **Apply** 应用图像的修改。
- **Help 按钮** 打开该面板的帮助文档。
- **Add 按钮** 添加新的图像或图标。
- **Delete 按钮** 删除当前选择的图标。
- **Clear 按钮** 清除该窗口中所有的图标。
- **Export 按钮** 将当前选择的所有图标保存到一个图像文件中。

## 2. 使用 **TImage** 组件设置背景图像

有时需要在表单中设置一幅背景图像，这样界面看起来会更加美观、生动。此时可以通过 **Additional** 面板的 **TImage** 组件实现。

设置背景的步骤如下：

(1) 通过 **TImage** 组件在表单中添加一个图像组件 **Image1**，然后将其 **Align** 属性设置为 **alClient**，这样图像框架就会铺满表单中的所有可显示区域。

(2) 单击 **Image1** 的 **Picture** 属性右边的按钮，则可以图像编辑器，然后单击 **Load** 按钮，则会打开文件选择窗口，从目录中选择合适的图像文件后再单击“打开”按钮，则又会返回到图像编辑器中，此时其显示如图 11-8 所示。

(3) 单击图像编辑器的 **OK** 按钮，则打开的图像会插入到 **Image1** 中。此时图像会按照其实际的大小进行显示。但是作为背景图像，一般希望其能够铺满表单的整个客户区域，此时只要将其 **Stretch** 属性设置为 **True** 即可。此时图像会铺满 **Image1** 的整个框架，同样也就铺满整个客户区域。

## 3. 绘制图形

绘制图形可以使用两种方法：一种是在设计阶段通过 **TShape** 组件绘制；其次是在运行阶段通过图形绘制函数绘制。





图 11-8 打开一幅图像时的图像编辑器

#### (1) 通过 TShape 组件绘制图形

其绘制步骤如下:

- 1) 在 Additional 组件面板选择 TShape 组件, 然后在表单中添加一个图形 Shape1。
- 2) 在 Shape 属性中选择合适的图形属性。这些属性的意义是: stCircle 表示圆形、stEllipse 表示椭圆形、stRectangle 表示矩形、stRoundRect 表示圆角的矩形、stRoundSquare 表示圆角的正方形、stSquare 表示正方形。可通过拖拉改变这些图形的大小及形状。
- 3) 通过 Brush 属性可以设置图形的填充颜色及类型。
- 4) 通过 Pen 属性, 可以设置图形边框的颜色及线条的类型。

#### (2) 运行时绘制图形

Delphi 提供了一个绘制图形的对象 TCanvas, 它提供了大量的属性和方法用来绘制图形。应用程序运行时绘制的图形都是通过该方法实现的。可以在表单对象 TForm、面板 TPanel、打印机对象 TPrinter、TPaintBox 上面绘制图形。

比如, 下面的代码在单击按钮 Button1 后, 在表单上绘制一条直线, 然后输出一行字:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Canvas.Pen.Color := clRed;           //设置绘制直线的颜色
    Canvas.MoveTo( 20, 50 );             //直线的起点
    Canvas.LineTo( 100, 200 );           //直线的终点
    Canvas.Font.Name := '隶书';           //下面3个语句设置字体的名称、大小及颜色
    Canvas.Font.Size := 20;
    Canvas.Font.Color := clBlue;
    Canvas.TextOut( Canvas.PenPos.x, Canvas.PenPos.y, '绘制了一条直线!' ); //
    绘制文字
end;
```

当然, 也可以在运行时绘制一幅位图图像, 此时需要使用 TBitmap 对象。比如, 下面的代码在单击按钮 Button1 后, 会绘制一幅设定的图像:

```
procedure TForm1.Button2Click(Sender: TObject);
```



```

var
  Bitmap : TBitmap;
begin
  Bitmap := TBitmap.Create;           //建立一个TBitmap实例
  try
    with Bitmap do begin
      LoadFromFile('d:\borland6\Borland
Shared\Images\Splash\256color\factory.bmp');
      Form1.Canvas.Draw(200,0,Bitmap);
    end;
  finally
    Bitmap.Free;
  end;
end;

```

上面的两部分代码执行的结果如图 11-9 所示。



图 11-9 绘制图形和图像

#### 11.4.2 添加多媒体功能

Delphi 提供了两个多媒体组件为 TAnimate 和 TMediaPlayer, 可用于建立一些多媒体应用程序, 比如播放声音、动画等。但是这些组件只能用于 Windows 应用程序中, 不能用于跨平台的应用程序中, 因为没有对应的 CLX 组件。

##### 1. 使用 TAnimate 组件

该组件只能显示一些 AVI 文件, 其关键属性是 FileName、Active、CommonAVI、StartFrame、StopFrame, 这些属性的意义如下:

- **FileName** 设置一个 AVI 文件, 以便在应用程序中播放该动画。
- **CommonAVI** 在表单中插入一些通用的 AVI 文件, 比如拷贝文件、搜索文件、删除文件等对应的动画。该属性只有设置为 aviNone, 才能使用 FileName 中设置的动画。如果设置其他的属性值, 则 FileName 属性中设置的动画文件将无效。
- **Active** 只有将该属性设置为 True, 才能播放 CommonAVI 和 FileName 属性中设置的动画文件。
- **StartFrame** 设置动画从哪一帧开始播放。
- **StopFrame** 设置动画停止在哪一帧。

比如，如果在 FileName 属性中设置 Delphi6 提供的 AVI 文件：

D:\Borland6\Delphi6\Demos\CoolStuf\cool.avi

则将 Active 属性设置为 True 时，在应用程序设计过程中就可以播放该动画。

## 2. 使用 TMediaPlayer 组件

TMediaPlayer 组件在 System 面板上，它会在表单中加入一个多媒体控制条，与录音机上的按钮非常相似，用于控制媒体的播放。TMediaPlayer 组件可以控制大部分的多媒体文件或设备，比如声音、动画、CD、录像机等。

TMediaPlayer 组件各个按钮的作用，如图 11-10 所示。

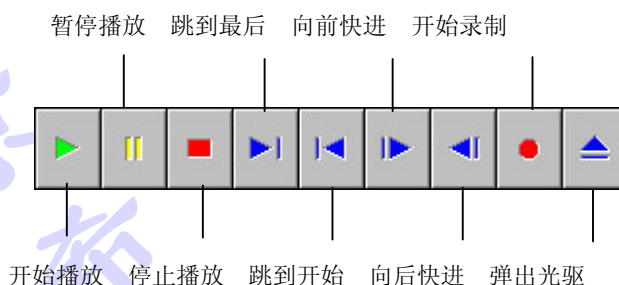


图 11-10 TMediaPlayer 组件各个按钮的作用

## 1. TMediaPlayer 组件主要的属性

主要属性如下：

- AutoOpen 该属性决定 MediaPlayer1 的播放按钮是否可用，只有设置为 True 时才可用。
- AutoRewind 该属性决定是否重复播放多媒体内容，属性值为 True 时表示重复播放。
- DeviceType 设置当前使用多媒体设备类型，默认值 DtAutoSelect 表示根据播放的文件类型决定使用的多媒体设备。各个选项的意义如下：
  - ◇ DtAutoSelect 该选项表示设备类型根据使用的多媒体文件类型决定，这些文件类型在 Windows 的 SYSTEM.INI 中定义。
  - ◇ DtAVIVideo Windows 的视频剪辑，扩展名是.avi。
  - ◇ DtCDAudio CD 播放机。
  - ◇ DtDAT 音频播放机。
  - ◇ DtDigitalVideo 数字电视。
  - ◇ DtMMMovie 电影放映机。
  - ◇ DtOther 其他媒体设备。
  - ◇ DtOverlay 模拟电视机。
  - ◇ DtScanner 扫描仪。
  - ◇ DtSequencer MIDI 合成器。
  - ◇ DtVCR 录像机。
  - ◇ DtVideodisc 视频播放机。
  - ◇ DtWaveAudio 数字化波形音频播放器。

- **Display** 决定播放动画、视频剪辑的位置或窗口，不设置该属性时会使用默认窗口。
- **Enabled** 决定 MediaPlayer1 组件上的播放按钮是否可用。
- **EnabledButton** 用于确定具体哪一个播放按钮可以使用。
- **FileName** 设置一个多媒体文件作为播放的媒体源。
- **VisibleButtons** 该属性具体决定哪一个按钮不可见。
- **DisplayRect** 将播放的动画、视频等内容限定在该属性设定的矩形框内。
- **StartPos** 设置开始播放多媒体的帧数或轨道数。
- **EndPos** 设置结束播放多媒体文件的帧数或轨道数。

## 2. 使用 TMediaPlayer 组件播放声音文件

步骤如下：

- (1) 在表单中添加一个媒体播放组件 MediaPlayer1。
- (2) 打开 MediaPlayer1 对应的对象观察器的属性面板，单击 FileName 属性右边的按钮，从目录中选择一个声音文件并确认，比如选择文件 c:\Windows\Media\Chord.wav。
- (3) 将 MediaPlayer1 的 AutoOpen 属性设置为 True。
- (4) 运行程序后，单击播放按钮，则会播放该声音文件。

## 3. 使用 TMediaPlayer 组件播放动画文件

其实现步骤与播放声音文件的步骤相同，只是在 FileName 属性中设置一个动画文件的名称即可，比如为：D:\Borland6\Delphi6\Demos\CoolStuf\cool.avi。

## 4. 隐藏播放按钮时播放动画

仍使用前面的 cool.avi 文件，然后将 MediaPlayer1 的 Visible 属性设置为 False，通过一个触发事件来播放动画，比如通过定时器控制定时播放动画。另外，通过设定 Display 和 DisplayRect 属性来限定播放动画的区域。其实现代码如下：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    MediaPlayer1.Open;                //打开媒体播放器
    MediaPlayer1.Display:=Form1;      //使动画显示在表单Form1中
    MediaPlayer1.DisplayRect:=Rect(0,0,100,50); //使动画显示在Form1的矩形区域中
    MediaPlayer1.Play;                //播放动画，此时AutoOpen属性设置为True
end;
```

**注意：**在播放多媒体文件时，控制焦点会转移到该多媒体文件上，所以会影响其他的操作。

## 11.5 使用对话框

Dialogs 组件面板提供了开发应用程序时常用的一些对话框，CLX 库中只提供了 6 个对话框，VCL 库则提供了更多的对话框。所有对话框的使用方法都非常相似，都是通过 Execute 方法打开，然后再获得一个返回值来设置其他对象。

下面分别通过一些实例说明这些对话框的使用方法。

### 1. 打开文件选择对话框 TOpenDialog

使用该对话框的步骤如下：

(1) 新建一个应用程序。

(2) 在表单中添加一个按钮 `Button1`、一个编辑框 `Edit1`、一个打开文件对话框 `OpenDialog1`。

(3) 双击按钮 `Button1`，然后在打开的代码编辑器中输入如下代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenDialog1.Execute;    //运行程序时，单击按钮Button1会打开文件选择框
end;
```

(4) 当选择一个文件并确认后，则会关闭文件选择对话框，此时该对话框会将选择的文件保存到 `FileName` 属性中，所以可以通过该属性获得选择的文件名。可以通过 `OpenDialog1` 的 `OnCanClose` 事件来获得该文件名，并在一个文本编辑框 `Edit1` 中显示出来，假设打开的是一个文本文件，则可以将显示在多文本组件 `Memo1` 中。实现该功能的代码如下：

```
procedure TForm1.OpenDialog1CanClose(Sender: TObject; var CanClose:
Boolean);
begin
    Edit1.Text :=OpenDialog1.FileName ; //在文本框中显示选择的文件名
    Memo1.Lines.LoadFromFile(edit1.Text);
end;
```

**注意：**`OpenDialog1` 有两个关闭窗口的事件，分别是 `OnCanClose` 和 `OnClose`，前者是没有使用“取消”按钮时关闭文件选择窗口，这样就能保证正确获得选择的文件名；而 `OnClose` 包括使用“取消”按钮关闭文件选择窗口，所以就无法保证获得选择的文件名。

另外，如果要筛选文件，可以单击 `Filter` 属性右边的按钮打开一个文本编辑器，然后将筛选的文件名以先后顺序排列，文件名都使用匹配符“\*”表示，后面使用文件扩展名进行限制。比如，设置了 3 个筛选条件时，该文本编辑框的显示如图 11-11 所示。单击 `OK` 按钮关闭该文件编辑框后，`Filter` 属性的值如下：

```
*.pas|*.dcu|*.*
```

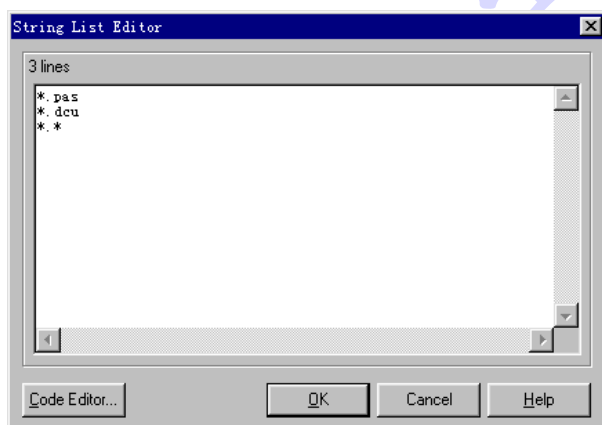


图 11-11 设置三个筛选条件

## 2. 使用保存文件对话框 `TSaveDialog`

TSaveDialog 与 TOpenDialog 对话框的使用方法相似, 我们通过按钮 Button1 打开保存文件对话框。当单击对话框的“保存”按钮关闭对话框时, 会触发 OnCanClose 事件, 通过该事件将 Memo1 中的内容保存到一个文本文件中, 并将该文本文件的名称显示在 Edit1 中。代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    SaveDialog1.Execute;
end;

procedure TForm1.SaveDialog1CanClose(Sender: TObject;
    var CanClose: Boolean);
begin
    edit1.Text := SaveDialog1.FileName;
    Memo1.Lines.SaveToFile(edit1.Text);
end;
```

### 3. 使用字体选择对话框 TFontDialog

字体选择对话框组件 TFontDialog 也是通过 Execute 方法打开, 该对象设置的字体保存在 Font 属性中。该对象没有任何事件, 所以, 可以直接在打开该对话框的事件中使用它设置的字体。比如, 下面的代码将字体对话框中选择的字体设置为标签 Label1 的字体:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    FontDialog1.Execute;
    Label1.Font := FontDialog1.Font;
end;
```

### 4. 使用颜色选择对话框 TColorDialog

颜色选择对话框的使用方法 & 功能与字体对话框 TFontDialog 非常相似, 可以通过 Execute 方法打开该对话框, 然后直接设置字体的颜色即可。通过一个按钮设置 Label1 字体颜色的代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ColorDialog1.Execute;
    Label1.Font.Color := ColorDialog1.Color;
end;
```

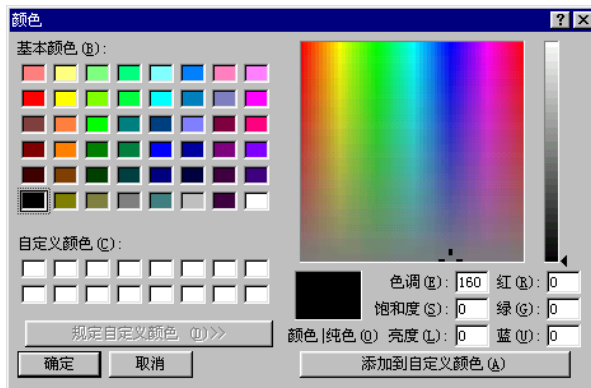


图 11-12 打开的颜色对话框

颜色对话框打开后的显示界面,如图 11-12 所示。左边小方格中是一些基本颜色,可以直接使用,也可以重新从右边的渐变色中选择合适的颜色。最右边的长条决定颜色的饱和度与亮度,可以拖拉右边的三角按钮来改变颜色的饱和度与亮度。

#### 5. 查询字符串对话框 TFindDialog

该对话框也需要通过 `Execute` 方法打开,可以通过一个按钮打开该对话框。然后通过 `TFindDialog` 组件的 `OnFind` 事件实现特定内容的搜索。下面的代码在 `RichEdit1` 中搜索符合条件的内容。由于可能有多个符合条件的内容,所以在搜索到符合条件的内容时,都要选择这些内容,以便突出显示;再查找下一个符合条件的内容时,应从当前选择的内容开始继续查找。`TFindDialog` 组件的 `OnFind` 事件在用户单击了查询对话框的“查找下一个”按钮后触发。完整的实现代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
    FoundPos: LongInt;
    StartPos, EndPos: Integer;
begin
    if RichEdit1.SelLength <> 0 then
        StartPos := RichEdit1.SelStart + RichEdit1.SelLength //从当前选择位置查找
    else
        StartPos := 0; //从头开始查找

    EndPos := Length(RichEdit1.Text) - StartPos; // 搜寻结束的长度
    FoundPos := RichEdit1.FindText(FindDialog1.FindText, StartPos, EndPos,
[stMatchCase]);
    if FoundPos <> -1 then //如果有符合条件的内容
    begin
        RichEdit1.SetFocus;
        RichEdit1.SelStart := FoundPos;
        RichEdit1.SelLength := Length(FindDialog1.FindText); //选择查找的内容
    end;
end;
```

查找对话框的显示如图 11-13 所示。



图 11-13 查找对话框的显示



注意：由于 CLX 库中没有提供 TRichEdit 组件，所以以上程序要使用 VCL 库实现。

#### 6. 查找并替换文本对话框 TReplaceDialog

TReplaceDialog 组件是在 TFindDialog 组件的基础上增加了一项新的功能，即使用新的内容替换查找到的字符串。它也需要通过 Execute 方法打开，然后通过 OnFind 事件来查找符合条件的字符串，所以这两个事件中的代码与 TFindDialog 组件对应事件中的代码相似，只要将 FindDialog1 替换为 ReplaceDialog1 即可，其他内容不需要修改。

要替换查找的字符串，还需要添加一个 OnReplace 事件，该事件在用户单击替换对话框中的“替换”按钮时触发。这三个事件的完整代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ReplaceDialog1.Execute;
end;

procedure TForm1.ReplaceDialog1Find(Sender: TObject);
var
    FoundPos: LongInt;
    StartPos, EndPos: Integer;
begin
    if RichEdit1.SelLength <> 0 then
        StartPos := RichEdit1.SelStart + RichEdit1.SelLength //从当前选择位置查找
    else
        StartPos := 0; //从头开始查找

    EndPos := Length(RichEdit1.Text) - StartPos; // 搜寻结束的长度
    FoundPos := RichEdit1.FindText(ReplaceDialog1.FindText, StartPos,
    EndPos,
        [stMatchCase]);
    if FoundPos <> -1 then //如果有符合条件的内容
    begin
        RichEdit1.SetFocus;
        RichEdit1.SelStart := FoundPos;
        RichEdit1.SelLength := Length(ReplaceDialog1.FindText); //选择查找的内容
    end;
end;

procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
    StartPos: Integer;
begin
    StartPos := Pos(ReplaceDialog1.FindText, RichEdit1.Lines.Text); //确定查找文本位置
    if StartPos > 0 then
    begin
        RichEdit1.SelStart := StartPos - 1;
        RichEdit1.SelLength := Length(ReplaceDialog1.FindText); //查找文本的长度
    end;
end;
```

```

RichEdit1.SelText :=ReplaceDialog1.ReplaceText; //替换查找的文本
end
else MessageDlg(Concat('在RichEdit1中没有找到"',ReplaceDialog1.FindText,
''),
mtError, [mbOk], 0); //弹出一个对话框, 报告错误信息
end;

```

TForm1.ReplaceDialog1Replace 过程中的 SelStart、SelLength、SelText 是 RichEdit1 的三个属性, 分别表示当前选择文本的开始位置、长度、文本内容; Pos、Length、Concat 则是三个字符串函数, 分别表示字符串的起始位置、长度及用于连接的字符串。

## 11.6 使用对象库

Delphi 提供了一个对象库, 可通过 File|New|Other 菜单打开它, 其中保存了各种类型的应用程序向导及实现特定功能的对象, 可用于快速建立应用程序、实现某些特定功能; 同时, 也可以将已经开发的应用程序、表单或数据模块保存到对象库中, 以便可以被一个开发组的所有人员共享, 从而大大提高工作效率。

VCL 及 CLX 应用程序对应的对象库是不同的。如果当前新建或打开的是一个使用 VCL 库的 Windows 应用程序, 则打开的对象库只能用于开发 Windows 平台的应用程序; 如果新建或打开的是一个使用 CLX 库的跨平台的应用程序, 则打开的对象库可用于开发跨平台的应用程序。如果当前没有新建或打开任何应用程序, 而直接打开对象库, 则该对象库中的内容合并了 VCL 及 CLX 两个对象库的内容。由于此时比较难于区分哪些用于 CLX, 哪些用于 VCL, 所以建议先新建或打开一个应用程序后, 再使用对象库中的内容。

### 11.6.1 对象库的类别

对象库按照类别划分为多个标签页, VCL 与 CLX 标签页中的内容有所不同。要使用 CLX 对象库, 应先通过 File|New|CLX Application 菜单新建一个应用程序, 然后通过 File|New|Other 菜单打开对象库, 此时的显示如图 11-14 所示。

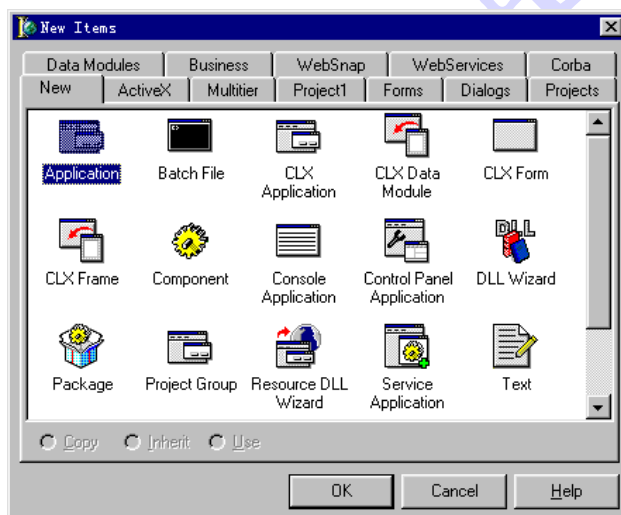


图 11-14 跨平台的应用程序对应的对象库

如果当前打开的是合并的对象库，则可通过其中一些对象建立 Windows 应用程序，通过另一些对象建立跨平台的应用程序，在实际应用中最好区别对待。合并的对象中各个标签页及其包含对象的作用如下：

- New 标签页 该页面包括了许多内置对象，可以直接应用到应用程序中。
  - ◇ Application 建立一个包含一个表单、单元文件和.dpr 文件等内容的应用程序。
  - ◇ Batch file 建立一个批处理项目文件，可以输入批命令，并使用.bat 扩展名保存，它不使用表单或代码编辑器。
  - ◇ CLX Application 建立一个包含一个表单、单元文件和.dpr 文件等内容的跨平台应用程序。
  - ◇ Component 使用组件向导建立一个新的组件。
  - ◇ Console Application 建立一个新的控制台应用程序。
  - ◇ Control Panel Application 建立一个用于 Windows 控制面板的 Applet 小程序。
  - ◇ Control Panel Module 建立一个新的控制面板应用程序模型。
  - ◇ Data Module 建立一个新的数据模块和对应的单元文件。
  - ◇ DLL 建立一个动态库项目文件，并以.dll 扩展名保存。
  - ◇ Form 向当前项目中添加一个新的表单。
  - ◇ Frame 建立一个新的框架。
  - ◇ Package 建立一个新的包。
  - ◇ Project Group 建立一个新的项目组来包含相关项目，它使用的文件扩展名为.bpg。
  - ◇ Report 用于建立一个快速报表，也可以通过 Business 标签页的 QuickReport 向导实现。
  - ◇ Resource DLL wizard 用于建立资源动态库。
  - ◇ Service 向 NT 控制面板的“服务”程序中添加一个新的服务。
  - ◇ Service Application 建立一个新的 NT 服务程序。
  - ◇ Text 建立一个新的 ASCII 文本文件。
  - ◇ Thread Object 建立一个新的线索对象。
  - ◇ Unit 向当前的动态库项目中添加一个新的单元文件。
  - ◇ Web Server Application 建立一个新的 Web Server 应用程序。
  - ◇ XML Data Binding 用于产生一个类来表示一个特殊的 XML 文档。
- ActiveX 标签页 该标签页的对象用于建立新的 COM 对象、Active 表单、ActiveX 控件、ActiveX 控件的属性页和类库等。各个对象的作用如下：
  - ◇ Active Form 建立一个新的 Active 表单。
  - ◇ Active Server Object 建立一个 ASP 程序。
  - ◇ ActiveX Control 建立一个新的 ActiveX 控件。
  - ◇ ActiveX Library 建立一个新的 ActiveX 库。

- ◇ Automation Object 建立一个新的 Automation (自动) 对象。
- ◇ COM Object 建立一个新的 COM 对象。
- ◇ COM+ Event Object 建立一个新的 COM+事件对象来向所有注册客户发送服务器事件。
- ◇ Property Page 建立一个 ActiveX 属性页面文件。
- ◇ Transactional Object 利用 MTS 或 COM+提供的分布式应用程序服务的优点来建立一个新的 Automation 对象。
- ◇ Type Library 建立或编辑一个类型库。
- Multitier 标签页 该标签页的对象用于建立多层应用程序中的服务器。
- ◇ CORBA Data Module 建立一个 CORBA 数据模块作为多层数据库应用程序中的一个服务器, 该应用程序使用的通信协议是 CORBA。
- ◇ CORBA Object 建立一个 CORBA 服务器对象。
- ◇ Remote Data Module 在基于 COM 的多层数据库应用程序中建立一个应用程序服务器。
- ◇ Transactional Data Module 建立一个利用 MTS 或 COM+服务优点的远程数据模块。
- Project1 标签页 可用于保存自己建立的对象。
- Forms 标签页 用于向应用程序中添加一种新的表单类型。比如 About 对话框、主从表数据、CLX 多标签页、快速报表等。
- Dialogs 标签页 提供建立各种对象框的向导, 比如建立带有帮助按钮的对话框、口令注册对话框、标准对话框等。
- Projects 标签页 提供建立一些应用程序的向导, 比如建立单文档 (SDI) 应用程序、多文档 (MDI) 应用程序、Windows 2000 或 Windows 9x Logo 应用程序。
- DataModules 标签页 用于建立不同类型的数据模型。
- Business 标签页 提供了用于商业方面的一些向导, 比如建立快速报表、数据库表单、数据库 Web 应用程序、直方图或饼图等向导。
- WebSnap 标签页 提供了建立 WebSnap 应用程序或 WebSnap 数据模块的一些向导。
- WebServices 标签页 用于建立一些 Web 应用程序。比如 Soap 服务器应用程序、Soap 服务器数据模块、Web 服务引入器等。
- Corba 标签页 用于建立 CORBA 客户和服务器应用程序。

#### 11.6.2 保存新对象到对象库中

前面说过, Delphi 应用程序、已经建立的表单及数据模块可以保存到对象库中, 以便以后重复使用相同的内容。要将一个应用程序保存到对象库中, 可以使用 Project|Add to Repository 菜单; 要将一个表单或数据模块保存到对象库中, 则应在该表单或数据模块上面单击鼠标右键, 从弹出的上下文菜单中选择 Add to Repository 菜单。在打开的对话框中设置保存该对象的标签页、对象的名称、使用的图标等属性, 确认后就可以保存到对象库中。

下面通过一个例子说明如何将一个表单保存到对象库中。

该实例的实现步骤如下：

1. 新建一个应用程序。
2. 在表单中添加一个数据源组件 DataSource1、基表组件 Table1、数据表格组件 DBGrid1 和数据导航组件 DBNavigator1。
3. 设置组件的属性，将 Table1 的 DatabaseName 属性设置为 DBDEMOS；将 DataSource1 的 DataSet 属性设置为 Table1；将 DBGrid1 和 DBNavigator1 组件的 DataSource 属性设置为 DataSource1。此时，表单 Form1 的显示如图 11-15 所示。

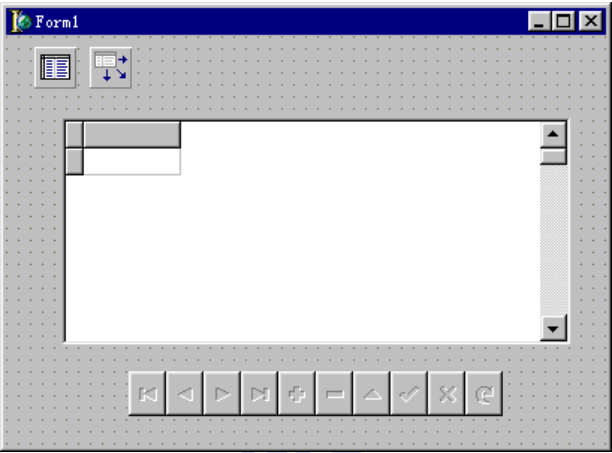


图 11-15 准备保存到对象库中的表单

4. 在表单 Form1 上面单击鼠标右键，从弹出的上下文菜单中选择 Add to Repository 菜单，则会弹出一个对象属性设置对话框。该对话框的设置如下：在 Title（标题）中设置名字为 my\_db\_form；在 Description（说明）编辑框中输入说明性内容；从 Page（页面）中选择对象库的 Forms 标签页；在 Author（作者）一栏输入作者的名字；最后通过 Browse 按钮为该对象选择一个图标。此时，该对话框的显示如图 11-16 所示。



图 11-16 设置将要保存到项目库中的对象属性

5. 单击对话框的 OK 按钮, 此时会弹出一个对话框, 如图 11-17 所示, 要求在添加到对象库以前, 先保存表单对应的单元文件。单击 Yes 按钮, 将其保存到磁盘中。最好将其保存到一个共享目录中, 以便共享。

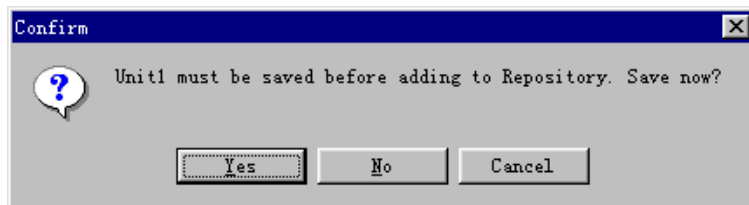


图 11-17 提示保存单元文件的对话框

6. 现在如果关闭整个应用程序, 然后重新建立一个应用程序, 再选择 File|New|Other 菜单, 并从 Forms 标签页中选择新建立的表单对象 my\_db\_form, 则会在应用程序中添加该对象对应的表单, 同时各个组件的属性也保持以前的设置值。

### 11.6.3 管理对象库

如果要修改或删除已经加入的对象, 如何才能实现? 这需要通过对象库属性面板实现。可以通过两种方法打开对象库属性面板: 一种是选择 Tool|Repository 菜单; 另一种方法是先通过 File|New|Other 菜单打开对象库, 然后单击鼠标右键, 从弹出的上下文菜单中选择 Properties 菜单。

在对象库属性面板中, 分别在 Pages 一栏选择 Forms, 在 Objects 一栏选择 my\_db\_form, 此时该面板的显示如图 11-18 所示。

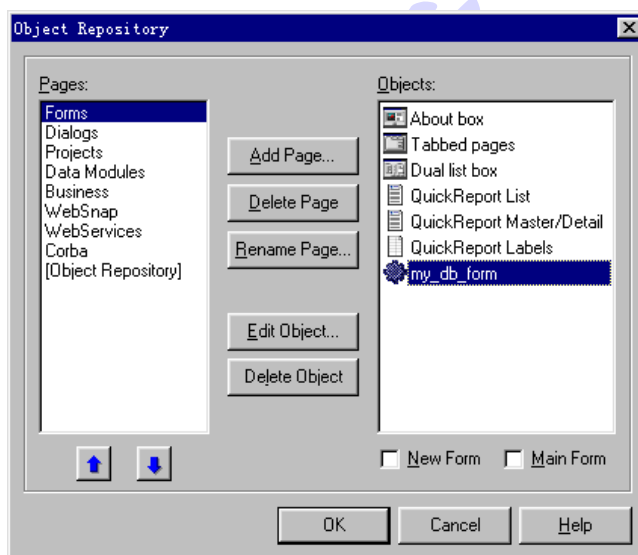


图 11-18 对象库属性面板

该面板有两栏: 左边 Pages 一栏列出了对象库中所有的标签页; 右边的 Objects 一栏列出了当前选择标签页中包含的对象。中间有 5 个按钮, 上面 3 个按钮用于标签页; 下面的 2 个按钮用于编辑或删除对象。这 5 个按钮的作用如下:

- **Add Page** 用于添加一个新的页面，可以在打开的对话框中输入一个新的标签页名字。
- **Delete Page** 用于删除当前选择的页面。
- **Rename Page** 用于重命名当前选择的页面。
- **Edit Objects** 编辑在 **Objects** 一栏当前选择对象的属性。
- **Delete Objects** 删除在 **Objects** 一栏当前选择的对象。

另外，**Pages** 一栏下面的上、下箭头按钮用于改变标签页的排列顺序；**New Form** 选项表示以后每次在应用中新建立一个表单时，默认都会使用该对象创建新表单；**Main Form** 选项会将当前选择的表单自动作为应用程序的主表单。另外，如果当前选择的对象是一个项目文件，还会显示 **New Project** 选项，该选项表示以后建立新项目时默认使用该对象。

**注意：**保存或修改对象属性时，尽量不要选择 **New Form**、**Main Form**、**New Project** 等选项，因为它们会使以前 Delphi 默认设置的表单及应用程序失效，而启用选择了这些选项的对象。如果已经选择了这些选项，则可在不需要时重新取消选择即可。

可以通过多种方式使用对象库中的对象，如果选择对象库中的 **Copy** 选项（这些选项见图 11-14），则会建立选择对象的一个副本；如果选择 **Inherit** 选项，则表示新对象会继承该对象；如果选择 **Use**，则表示直接使用该对象。



## 第 12 章 处理应用程序的异常

开发应用程序时，难免会产生各种错误，存在许多漏洞，从而导致最终运行的应用程序显示一些错误信息甚至导致系统崩溃。Delphi 提供了一种良好的系统保护机制，可以在应用程序运行过程中处理这些错误，从而避免系统崩溃，这种保护机制称为异常处理。简单地说，异常是处理应用程序错误的一种机制，所以它与错误本身表示的意义不一样。

### 12.1 定义异常

在应用程序中要处理一个异常，必须提前定义这些异常。Delphi 已经定义了许多最常遇到的异常，在应用程序中可以直接处理这些异常；另外，也可以在应用程序中单独定义一些异常，然后对这些异常进行处理。

#### 12.1.1 异常的基类 Exception

Delphi 中所有异常的基类是 Exception 对象，它直接继承自所有组件的祖先 TObject 对象，该对象在 sysutils 库单元中定义，它封装了所有异常的基本属性和方法，可以在该对象基础上建立自定义的异常。

来自 Exception 对象的所有方法可以作为建立异常消息的一种构造器，一些构造器也建立帮助内容的 ID。通常，一个应用程序可以在出现一个异常时动态调用这些构造器。异常消息可以视察硬编码的字符串、格式化的字符串或来自一个应用程序资源的字符串（包括格式化字符串）。在一个应用程序中遇到运行错误时会提交异常，比如企图将一个字母组成的字符串转换为数字值时。在提交一个异常时，通常会显示一个对话框来描述导致错误的原因。如果一个应用程序不能处理异常，则会调用默认的异常处理器，该处理器也会显示一个带有 OK 按钮的对话框，在用户单击该 OK 按钮后，应用程序可以继续执行。

Exception 对象为处理错误提供了一个一致的接口，能够使应用程序以一种恰当的方式处理错误，应用程序可以使用 try...except 语句块来处理特定的异常；也可以通过 try...finally 语句块来保护代码的正常执行及资源的合理分配；还可以通过使用 Raise 语句在遇到异常时提交该异常。

#### 1. Exception 对象的属性

Exception 有 HelpContext 和 Message 两个属性，其他异常会继承这两个属性。

- HelpContext 属性 提交该异常帮助信息对应的序列号。该属性是一个 THelpContext 对象，它提供了所有异常对象的帮助信息对应的序列号。
- Message 属性 当 Delphi 应用程序遇到一个异常时，会弹出一个对话框来显示对当前错误的描述信息，该信息就是 Message 属性的值。

#### 2. Exception 对象主要的方法

- Create 方法 该方法用于创建一个异常的实例，通过该实例可以显示错误信息，还可以直接使用该方法来提交一个异常。其语法如下：

```
constructor Create (const Msg: string);
```

其中, **Msg** 参数包含的是一个用于描述错误信息的字符串。它可以是一个编码的字符串,也可以是一个字符串函数。

例如,下面的过程用于产生一条被 0 除的错误信息,最终界面及错误提示框如图 12-1 所示。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a,b:real;
begin
  try
    a:=0;
    b:=5/a;    //正常执行语句
    edit1.Text :=FloatToStr(b);
  except
    raise Exception.Create('错误,除数不能为0!'); //显示错误信息
  end;
end;
```

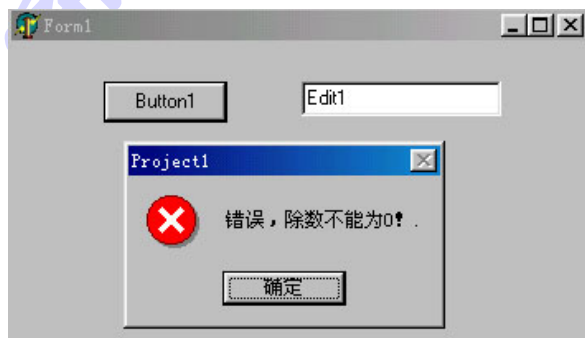


图 12-1 使用 Create 方法显示错误信息

- **CreateFmt** 方法 该方法与 **Create** 方法的作用相同,只是它将字符串信息进行了格式化,其语法如下:

```
constructor CreateFmt (const Msg: string; const Args: array of const);
```

**Args** 参数是个数组,它包含了格式化的字符串,使用全局函数 **Format** 来格式化字符串,格式由 **Msg** 参数决定,其最终的返回值将替代 **Msg** 参数中的值。

- **CreateFmtHelp** 方法 该方法可用于显示帮助信息及帮助序列号。其语法如下:

```
constructor CreateFmtHelp (const Msg: string; const Args: array of const;
  AHelpContext: Integer);
```

**Msg** 参数包含提示信息; **Args** 参数是一个常量数组,格式由 **Msg** 参数决定; **AHelpContext** 参数用于决定帮助信息的序列号。

- **CreateHelp** 方法 用于显示帮助信息和帮助序列号。其语法如下:

```
constructor CreateHelp (const Msg: string; AHelpContext: Integer);
```

其中 **Msg** 参数表示帮助信息, **AHelpContext** 参数代表了帮助信息的序列号。

- **CreateRes** 方法 显示的帮助信息来自一个资源文件或一个资源字符串。资源文件定义了应用程序使用的一些资源信息,应用程序编译时,需要捆绑资源文件。该方法的语法如下:

```
constructor CreateRes (Ident: Integer); overload;
```

```
或 constructor CreateRes (ResStringRec: PResStringRec); overload;
```

其中, Ident 参数表示资源的序列号; ResStringRec 参数是一个指向资源字符串的指针。

例如, 下面的语句用于建立资源错误提示信息, 其中符号@表示变量的地址:

```
resourcestring sMyNewErrorMessage = '这里是错误的资源。'
```

```
.....
```

```
Exception.CreateRes (@sMyNewErrorMessage);
```

### 12.1.2 Delphi 定义的异常

Delphi 在多个标准库中定义了一些比较通用的异常, 比如被 0 整除、类型转换错误、数据库引擎错误等。Windows 下的 VCL 库和跨平台的 CLX 库的名称非常相似, 定义的异常类型也比较相似。下面以 VCL 库为例列出一些比较通用的异常, 这些异常大部分在 Sysutils 库文件中定义, 可以在 Delphi 的子目录下打开该库文件查看这些文件的定义; 另一些异常在其他库中定义, 比如 db、adodb、dbtables 等多个库。

- EAbort 如果在应用程序遇到错误时不希望弹出错误消息对话框来处理该错误, 就可以使用该异常。
- EAccessViolation 当遇到无效的内存访问错误时, 会使用该异常。
- EAssertionFailed 当执行 Assert 过程的返回值是 False 时, 使用该异常。
- EControlC 在控制台应用程序中按下 Ctrl+C 键遇到错误时, 使用该异常。
- EConvertError 进行字符串或其他对象转换时遇到错误, 使用该异常。
- EDivByZero 如果整数的除数是 0, 使用该异常。
- EInOutError 遇到输入/输出错误时, 使用该异常。
- EIntOverflow 进行整数运算时, 如果结果超过了变量定义的范围, 使用该异常处理问题。
- EMathError 进行浮点数学运算出现错误时, 使用该异常。
- EOutOfMemory 遇到超出内存的错误时, 使用该异常。
- EOverflow 进行浮点运算返回的结果超过变量定义的范围时, 使用该异常。
- ERangeError 整数值超过变量定义的范围时, 使用该异常。
- EStackOverflow 堆栈溢出时, 使用该异常。
- EWin32Error 出现 Windows 错误时, 使用该异常。
- EZeroDivide 如果浮点数的除数是 0, 使用该异常。

注意: 以上所有异常在 Sysutils 库中定义。

- EDatabaseError 当遇到数据库错误时, 使用该异常, 该异常在 db 库中定义。
- EUpdateError 当数据集提供者出现错误时, 使用该异常, 该异常在 db 库中定义。
- EADOError ADO 数据库遇到错误时, 使用该异常, 该异常在 adodb 库中定义。
- EDBEngineError 如果出现 BDE 错误, 使用该异常, 该异常在 dbtables 库中定义。
- ENoResultSet 如果没有成功打开一个数据集的查询, 则使用该异常。

### 12.1.3 处理异常

Delphi 提供了一些函数或过程, 可以直接用来处理一些异常。在 VCL 及 CLX 库中分

别定义了用于 Windows 和 Linux 平台上的函数或过程，有许多函数或过程的名称及作用基本相同，下面以 CLX 库为例进行说明，因为 CLX 库既可以用于 Windows 平台，也可用于 Linux 平台。

- DatabaseError 调用该过程会建立或提交一个 EDatabaseError 异常，其语法如下：  
procedure DatabaseError(const Message: string; Component: TComponent = nil);
- GetLastError 返回操作系统 API 调用的最后一个错误，其语法如下：  
function GetLastError: Integer;
- OutOfMemoryError 该过程提交一个 EOutOfMemory 异常。
- RaiseLastOSError 该过程用于提交操作系统或系统库出现的错误导致的异常。
- ShowException 用于显示一条异常的消息及其物理地址。语法如下：  
procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer);
- SysErrorMessage 将操作系统错误代码转换为字符串。其语法如下：  
function SysErrorMessage(ErrorCode: Integer): string;

#### 12.1.4 自定义异常

由于应用程序在运行的过程中可能会出现各种各样的异常，所以 Delphi 已经定义的一些异常可能无法满足应用程序处理错误的要求，此时可以在应用程序中定义一些异常来处理这些错误。

1. 通过 Exception 对象定义异常。由于 Exception 对象是所有异常的祖先，所以其他异常都要通过 Exception 来定义。另外，也可以在已经定义的异常基础上再声明其他的异常。

例如，在 SysUtils 库中通过 Exception 声明一个数学运算异常的语句如下：

```
type
    EMathError = class (Exception);
end;
```

2. 通过 Exception 对象的子对象定义异常。比如，通过前面定义的 EMathError 异常可以声明其他的异常，例如：

```
type
    EMathError = class (Exception);
    EInvalidOp = class (EMathError);
    EZeroDivide = class (EMathError);
    EOverflow = class (EMathError);
    EUnderflow = class (EMathError);
end;
```

这里声明的 EInvalidOp、EZeroDivide、EOverflow、EUnderflow 等几种类型的异常都继承自 EMathError，所以这几种类型的异常都可以通过处理 EMathError 异常来处理它们。当然，也可以单独处理每个异常，这样程序代码更清晰。

3. 定义并提交异常来处理一般错误。在应用程序中可以通过定义异常来处理各种错误。比如，希望在用户输入错误的口令时提交一个异常，假设正确的口令保存在 CorrectPassword 字符串变量中，用户输入的口令保存到 Password 字符串变量中，则可以使用如下的代码来定义并提交一个口令输入错误的异常：

```
type
```

```

EPasswordInvalid = class(Exception);    //声明一个异常
.....
if Password <> CorrectPassword then
    raise EPasswordInvalid.Create('口令错误!');

```

### 12.1.5 捕获错误

有时,我们希望应用程序能够捕获对应某个对象的错误代码及对该错误的描述等信息,则可以通过该对象的 `Error` 和 `ErrorMessage` 属性来获得这些错误代码及信息,例如,下面的代码在打开多媒体播放器 `MediaPlayer1` 时,如果遇到异常,则会弹出一个窗口显示错误的代码及内容,代码如下:

```

procedure TForm1.BitBtn1Click(Sender: TObject);
var
    MyErrorString: String;
begin
    try
        MediaPlayer1.Open;
    except
        MyErrorString := '错误代码及错误内容如下: ' + IntToStr(Error) + #13#10;
        MessageDlg(MyErrorString + MediaPlayer1.ErrorMessage, mtError, [mbOk],
0);
    end;
end;

```

## 12.2 处 理 异 常

为了保证程序的正常运行,并能够合理地释放资源,在应用程序中不但要定义一些异常,同时还要处理这些异常,这样程序就能够继续执行。但是由于引起错误的原因众多,有时可能很难预测,所以还应该提供一种保护机制,使程序能够正常地运行。下面就分析一下 Delphi 提供的保护程序正常运行的措施及处理各种异常的方法。

### 12.2.1 try...finally 保护块

该保护块的作用如下:正常执行的代码放置在 `try` 到 `finally` 之间,这些代码会按照程序的执行顺序运行;一旦这些代码中出现了错误,则程序会跳转到 `finally` 后面的第一行代码,并按顺序开始执行后面的代码,这些代码用于处理可能遇到的各种错误,或者释放程序使用的资源后,关闭该应用程序的执行。如果 `try` 部分的代码没有遇到错误,在执行完该部分所有的代码后,仍会执行 `finally` 部分的代码,所以可以将释放资源的语句放置在该部分,这样程序的结构就比较清晰。

该代码块的结构如下:

```

...    //设置一些程序初始化语句
try
...//正常执行的代码
finally
    ...//处理错误或释放资源的代码
end;

```

Delphi 通过提供这样一种保护机制，使程序无论遇到何种错误，都能释放占用的系统资源，保证程序的正常退出，而不会引起系统崩溃或死机等现象。

当然，try...finally...end 代码保护块也可以进行嵌套，即在 try 和 finally 之间再加入一个 try...finally...end 代码保护块，以便对代码进行更详细的处理。

1. 第一个例子。下面的代码使用 try...finally...end 结构来释放文件使用的资源：

```
Reset (F);
try
    ...                //正常执行的代码
finally
    CloseFile (F);
end;
```

现在有一个问题，就是正常执行的代码中如果使用了 Break、Continue 或 Exit 等非顺序执行的语句时，程序会如何执行？是否会跳过 finally 后面的语句？答案是否定的。即无论 try 部分语句是否完全执行，程序都会执行 finally 后面的语句。

当然，如果 try...finally 之间的代码没有遇到某个异常，则不会执行 finally 后面对该异常处理的代码。

2. 第二个例子。下面对是否使用 try...finally...end 语句块保护程序导致的结果进行比较：

(1) 存在缺陷的代码。完整代码如下：

```
procedure TForm1.Button1Click(Sender: TComponent);
var
    Apointer: Pointer;
    AnInteger, Adividend: Integer;
begin
    Adividend := 0;
    GetMem(Apointer, 1024);           //分配内存
    AnInteger := 10 div Adividend;    //出现了一个被0除的错误
    FreeMem(Apointer, 1024);         //该释放内存的语句无法执行，因为出现了异常
end;
```

上面的代码遇到一个问题：因为遇到被 0 除的异常，所以可能会终止程序的运行，从而无法执行后面的 FreeMem 语句，也就无法释放该应用程序使用的内存。

要使应用程序出现错误时仍能释放内存，则应在应用程序中加入保护代码块。

(2) 正确的代码。纠正上面代码错误的方法是使用 try...finally...end 保护块，因为无论程序是否出现错误，都会执行 finally 后的代码，所以可以将释放内存的 FreeMem 语句放置在 finally...end 之间，正确的代码如下：

```
procedure TForm1.Button1Click(Sender: TComponent);
var
    Apointer: Pointer;
    AnInteger, Adividend: Integer;
begin
    Adividend := 0;
    GetMem(Apointer, 1024);
    try
        AnInteger := 10 div Adividend; //仍出现被0除的错误
    finally
        FreeMem(Apointer, 1024);
```



```

        FreeMem(APointer, 1024);    //合理地释放内存
    end;
end;

```

### 12.2.2 使用 try...except 处理异常

类似于 try...finally...end 程序块保护程序正常的执行一样，try...except...end 语句块通过对异常进行处理，也可以达到保护程序正常执行的目的。其正常执行的代码也放置在 try...except 之间，处理异常的代码则放置在 except...end 之间。

其完整的结构如下：

```

try
    ...           //正常执行的代码
except
    on ...do...
    on...do...
    ...           //多个处理异常的语句
end;

```

处理异常的代码通常以 on 开头，在 on 和 do 之间使用异常的名字，处理异常的代码放在 do 的后面。例如，要处理被 0 除时出现的异常，可以使用下面的代码：

```

try
    c: = a div b;
except
    on EZeroDivide do ShowMessage('出现除数为0的错误!');
end;

```

try...except...end 语句块处理完异常后，会重新跳转到出现异常的代码位置，并向下继续执行程序。

另外，可在 on 语句中直接使用异常的基类 Exception 来声明一个异常实例，用于显示一些异常信息或处理异常，例如，下面的代码通过定义一个异常实例来捕获错误信息及序列号：

```

try
    ...
except
    on E: Exception do ShowMessage('错误信息是: '+E.Message+#13#10+'错误序列号是: '+E.HelpContext);
end;

```

另外，可以在 except...end 之间使用 else 语句，并在 else 后面添加处理前面没有列出的异常，可以使用 HandleException 全局过程处理这些异常，例如：

```

try
    ...
except
    on EZeroDivide do HandleZeroDivide;
    on EOverflow do HandleOverflow;
    on EMathError do HandleMathError;
else
    HandleException;
end;

```

当然，类似于 try...finally...end 语句保护块，try...except...end 异常处理块也可以相互



嵌套, 嵌套代码块要放置在主代码块的 `try...except` 之间。另外, 它也可以与 `try...finally...end` 语句保护块进行嵌套, 嵌套代码块要放置在主代码块的 `try...finally` 之间。

### 12.2.3 使用 `raise` 处理异常

Delphi 提供了一种非常简捷的异常处理语句 `raise`, 通过该语句可以随时在正常执行的代码中加入处理异常的语句, 该语句需要提交一个异常的实例, 该实例可通过异常基类 `Exception` 的 `Create` 开头的一些方法建立, 也可以创建 Delphi 定义的其他异常实例。`raise` 语句的格式如下:

```
raise object at address
```

其中, `object` 和 `address` 是可选项, 如果省略 `Object`, 会提交当前的异常; 如果省略 `address`, 会使用一个过程或函数。

例如下面语句产生一个除数为 0 的异常实例:

```
raise EZeroDivide.Create;
```

在下面的实例中, 如果 `Result` 变量中的值超出了整型数的定义范围, 就会通过 `raise` 语句提交一个 `ERangeError` 异常, 然后可以继续执行其他代码。完整代码如下:

```
Procedure StrToIntRange(const S:string;Min, Max:Longint):Longint;
begin
    Result := StrToInt(S);
    if (Result < Min) or (Result > Max) then
        raise ERangeError.CreateFmt('%d超出变量定义的范围:%d..%d',[Result, Min,
        Max]);
    ...    //其他代码
end;
```

当然 `raise` 后面可以不使用任何对象, 此时仍会处理异常, 但只能进行较低程度的异常处理, 比如当一个过程或函数中的异常出现后清除, 但是还有其他异常需要处理时, 就可以只使用 `raise` 语句来处理。

例如, 下面的函数 `GetFileList` 中, 如果没有使用 `raise` 语句, 在出现初始化失败、路径不存在、内存不足等情况时会导致异常, 使应用程序无法正常继续执行, 此时用户根本无法知道出现了异常, 因为这类异常不太明显, 并且不会显示错误信息提示框。如果在 `try...except...end` 语句块中使用 `raise` 语句, 就可以处理这些可能出现的异常, 从而保证应用程序的正常执行。该函数完整的代码如下:

```
Function GetFileList(const Path:string):TStringList;
var
    I:Integer;
    SearchRec:TSearchRec;
begin
    Result := TStringList.Create;           //建立一个TStringList实例
    try
        I := FindFirst(Path, 0, SearchRec);    //查找路径
        while I = 0 do
            begin
                Result.Add(SearchRec.Name);      //将查找的结果添加到TStringList实例中
                I := FindNext(SearchRec);        //查找下一个符合条件的路径
            end;
    end;
```

```

except
    Result.Free;           //销毁TStringList实例
    raise;                 //处理可能出现的异常
end;
end;

```

使用 **raise** 语句时, 也可以实现多层异常处理语句块的嵌套, 其格式类似于下面的代码:

```

try
    ...{正常执行的代码}
try
    ...{特殊代码}
except
    on ESomething do      // ESomething表示一个异常的名字
    begin
        ... { 处理特殊的语句 }
        raise; { 提交遗漏的异常 }
    end;
end;
except
    on ESomething do ...; {处理其他异常}
end;

```

#### 12.2.4 使用外部资源处理异常

默认情况下, **HandleException** 方法会处理应用程序的异常, 但是在跨平台的应用程序中, 一般不需要调用 **TApplication.HandleException** 方法, 而当需要编写共享对象文件或回调函数时, 可以使用 **TApplication.HandleException** 来防止有异常出现在你的代码以外, 尤其是当一个不支持异常处理的外部资源调用这些代码时。

例如, 如果一个异常在应用程序代码中传递所有的 **try** 块, 则应用程序会自动调用 **HandleException** 方法, 当出现错误时会显示一个对话框。使用 **HandleException** 方法处理异常的格式如下:

```

try
    ...{正常执行的代码}
except
    Application.HandleException(Self);
end;

```

除了 **EAbort** 以外的所有异常, 如果有 **OnException** 可用, 则 **HandleException** 方法会调用该事件处理器。因此, 如果你希望既处理异常, 又将其作为内置组件的默认行为, 则可以按照下面的方法在你的代码中调用 **HandleException** 方法:

```

try
    ...{ 特殊语句 }
except
    on ESomething do
    begin
        { 只处理特殊语句 }
        Application.HandleException(Self);    {调用HandleException 方法}
    end;
end;

```

注意：不要在一个单线索的异常处理程序中调用 `HandleException` 方法。

### 12.2.5 处理哑异常

Delphi 提供了一种良好的异常处理机制，可以处理应用程序中大部分的异常，即使你的应用程序中没有提供处理这些异常的代码。此时在遇到异常时，应用程序通常会显示一个对话框，描述了导致异常出现的原因，关闭该对话框时，可以继续程序的执行。但是由于这些对话框通常是英文的，所以应尽量避免它们的出现，有没有办法屏蔽这些对话框而又不影响应用程序的正常执行呢？此时，可以定义一个哑（silent）异常来屏蔽该对话框。

所有的哑异常都继承自标准的异常类型 `EAbort`。Delphi 的 VCL 和 CLX 应用程序中，除了继承自 `EAbort` 的异常以外的所有默认异常处理器，都会显示一个错误信息对话框。注意，在控制台应用程序中，未处理的 `EAbort` 异常也会显示一个错误信息对话框，这是提交哑异常的捷径。调用 `Abort` 过程可以代替手工建立哑异常，因为该过程会自动提交一条 `EAbort` 异常，它会中断当前的操作，而不会显示错误消息对话框。

例如，下面的代码就是通过调用 `Abort` 过程来提交一个哑异常并中断当前的操作：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 10 do
  begin
    ListBox1.Items.Add(IntToStr(I));
    if I = 7 then Abort; //中断循环
  end;
end;
```

### 12.2.6 处理 RTL 异常

当在应用程序中调用了一些函数或过程，在程序运行过程中就可能出现一行异常，此时运行库（runtime library，RTL）一般会以一个对话框的形式显示这些错误信息。如果要屏蔽掉这些信息，或者希望用中文对话框的形式显示这些错误信息，则只要在应用程序中对可能出现的异常进行适当的处理即可。

RTL 异常与其他异常一样，也是继承自异常的基类 `Exception`，这些异常都是在 `SysUtils` 库中定义的。RTL 可以提交的异常类型，见表 12-1。

表 12-1 RTL 可以提交的异常类型

错误类型	导致错误的原因	说明
Input/output (输入/输出)	不正确地访问了一个文件或 I/O 设备	大多数 I/O 异常与访问文件时返回的错误代码有关
Heap (堆栈)	错误使用动态的内存	当没有足够的内存或者应用程序处理一个指向堆栈外的指针时导致该错误
Integer math (整数运算)	整型表达式中不合逻辑的操作	包括被 0 除、超出定义范围的数值或表达式、数据溢出等错误

(续表)

错误类型	导致错误的原因	说明
Floating-point math (浮点数学运算)	实型表达式不合逻辑的操作	该错误可能来自硬件处理器或软件仿真器的错误, 包括无效指令、被 0 除及数据的上溢和下溢等
Typecast (类型匹配)	as 操作符无效的类型匹配	对象间只能是兼容的类型进行匹配
Conversion (类型转换)	无效的类型转换	类似 IntToStr, StrToInt 的类型转换函数由于参数不正确引起的错误
Hardware (硬件)	系统的条件	硬件或用户产生的一些错误条件或中断, 比如违例访问、堆栈溢出、键盘中断等
Variant (变量)	不合逻辑类型的强制转换	当引用表达式的变量时, 该变量无法进行转换时导致该错误

可以通过下列方法处理 RTL 异常。

### 1. 建立异常处理器

异常处理器可以处理特定的异常或保护代码块中的异常。在跨平台的应用程序中, 很少需要手工编写异常处理代码, 因为大部分异常可以直接使用 `try..finally...end` 语句块进行处理。要定义一个异常处理器, 只要在 `try...except` 语句块 `except` 后的语句中, 加入要保护的程序或需要处理的异常即可。

应用程序运行时会先执行 `try` 部分的语句, 包括该部分调用的函数或过程在运行时遇到的异常。只有在 `try` 部分遇到异常时才会执行 `except` 后面的语句; 如果没有异常出现, 则不会执行这些语句。

当在 `try` 语句块中遇到异常时, 程序会立即跳转到 `except` 部分, 然后查找与当前出现的异常相对应的处理语句, 如果找到对应的异常处理语句则会执行这些语句, 在执行完该异常的处理语句后, 会立即销毁该异常实例, 同时程序重新跳转到原来出现异常的断点位置, 继续执行其他的语句。所以出现一个异常时, 只会运行与其对应异常处理语句, 而不会运行其他的异常处理语句, 从而保证应用程序可以高效地运行。如果出现了一个异常, 而在 `except` 部分没有找到相应的处理语句, 则将无法处理这些异常。所以在建立应用程序时, 应在 `except` 部分使用 `else` 语句, 然后通过 `HandleExcept` 或 `Abort` 方法来处理前面语句中可能遗漏的异常。

使用 `else` 语句时, 处理异常的语句块结构如下:

```
try
  ...{ 正常执行代码 }
except
  on ESomething do
    ...{ 处理指定的一种异常 };
  else
    ...{ 默认异常处理代码, 处理 on...do 中没有列出的异常 };
end;
```

由于 `else` 后面的默认异常处理代码会处理遗漏的所有异常, 所以这样将无法发现导致异常出现的原因, 此时可以使用下面的 `try...finally...end` 语句块来代替它:

```
try
  try
```

```

...{ 正常执行的代码 }
except
    on ESomething do {处理指定的一种异常};
end;
finally
    ...{ 释放资源的代码 };
end;

```

## 2. 使用异常实例

try...except...end 语句块的 except 部分对异常的处理,一般是通过 on...do 语句实现的。其结构如下:

```
on <type of exception> do <statement>
```

例如,下面的函数中通过 on...do 语句来处理被 0 除的异常:

```

function GetAverage(Sum, NumberOfItems: Integer): Integer;
begin
    try
        Result := Sum div NumberOfItems;    { 正常执行语句 }
    except
        on EDivByZero do Result := 0;    { 如果NumberOfItems为0时,才执行该语句 }
    end;
end;

```

在使用 on...do 语句的异常处理块中,一般情况下只需要异常的类型而不需要其他信息。

在异常处理块中要获得一个异常实例的其他信息,必须对 on...do 语句结构进行修改,一般通过定义一个临时变量来保存一个异常实例,然后再访问该异常实例。

例如,建立了一个应用程序,在其表单中加入一个滚动条和一个按钮,双击该按钮并在该按钮的 OnClick 事件中加入如下代码:

```
ScrollBar1.Max := ScrollBar1.Min - 1;
```

在应用程序执行时,该行代码就会提交一个异常,显示一个错误提示框如图 12-2 所示,报告的错误是 ScrollBar 设置的属性超出范围,因为滚动条的最大值不可能小于其最小值。

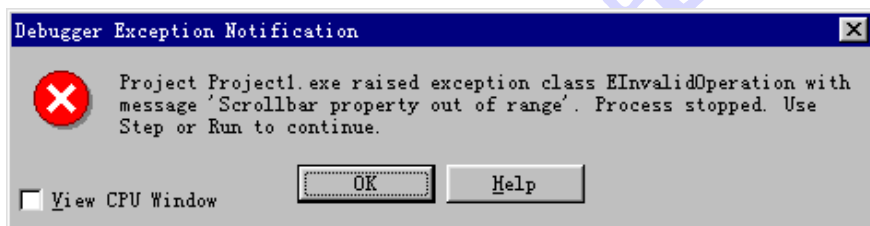


图 12-2 错误提示框

如果要屏蔽该错误提示框,而使用自己设置的错误提示框,就需要对这种异常进行处理,如下面的代码所示:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    try
        ScrollBar1.Max := ScrollBar1.Min - 1;
    except
        on E: EInvalidOperation do

```

```

        MessageDlg ('出现异常: ' + E.Message, mtInformation, [mbOK], 0);
    end;
end;

```

上面的代码中定义了一个临时变量 E（后面使用冒号），作为 `EInvalidOperation` 异常的一个实例。`E.Message` 表示通过异常实例 E 的 `Message` 属性来显示错误信息。另外，也可通过 `as` 操作符定义异常的实例。

**注意：**不要手工销毁代码中定义的异常实例，应用程序在使用完该实例后，会自动将其销毁。

### 3. 重新提交异常

当使用嵌套的异常处理块时，如果内层与外层有相同的异常处理代码，则只会执行内层的异常处理，同时在处理完异常后会销毁这些异常实例，而外层的异常处理语句将无法执行。但大多数情况下，用户希望了解更多的有关异常产生的信息，或者将这些错误信息保存到一个文件中，然后再进行标准的异常处理。此时可以使用 `raise` 语句，这样在处理完内层的异常后，会继续提交这些异常，然后由外层的异常处理语句来处理；这称为重新提交异常。其使用的结构如下：

```

    try
    ...{ 外层正常执行的语句 }
    try
    ...{ 内层正常执行的语句 }
    except
    on ESomething do
    begin
    ...{ 处理特定的异常 }
    raise; { 重新提交这些异常 }
    end;
    end;
except
on ESomething do ...; { 外层的异常处理语句 }
end;

```

如果外层的代码提交了一个异常，则只会执行外层 `except` 部分的异常处理语句。但是，如果在内层提交了一个异常，则首先会执行内层 `except` 部分的异常处理语句，然后再执行外层 `except` 部分的异常处理语句。

通过使用 `raise` 再次提交异常，可以保证所有的异常都会得到处理，同时可避免内外层异常处理的重复或遗漏。

#### 12.2.7 调试程序时处理异常

使用 `Tools|Debugger Options` 菜单可以打开调试器选项，其中的 `Language Exceptions`（语言异常）和 `OS Exceptions`（操作系统异常）两个标签页可以设置程序调试阶段如何处理异常，下面分别说明如何设置这些选项及它们的作用。

##### 1. Language Exceptions 标签页

该标签页的显示如图 12-3 所示。它包括如下选项和按钮：

- **Exception Types to Ignore**（忽略的异常类型） 下面的列表框中列出了在程序调试阶段可以被忽略的异常类型，如果选择了一种异常类型，则该类型在程序调试时

会被忽略，这样在调试程序时如果遇到这类异常，应用程序不会终止执行。还可以通过 Add 和 Remove 按钮，向列表中添加其他的异常类型。比如，如果向列表中添加了一个 EMathError 异常，并选择其对应的复选框，表示调试时忽略该异常。这样调试应用程序时，如果提交了该异常，则调试程序仍会正常执行，不会在出现该异常时终止执行。另外，如果一个异常继承自该异常，也会被忽略，比如，EOverflow 异常就继承自 EmathError，所以调试程序时遇到 EOverflow 异常，也不会终止程序的执行。

默认情况下，列表框中显示的异常类型、默认设置值和对应的异常名称，见表 12-3。

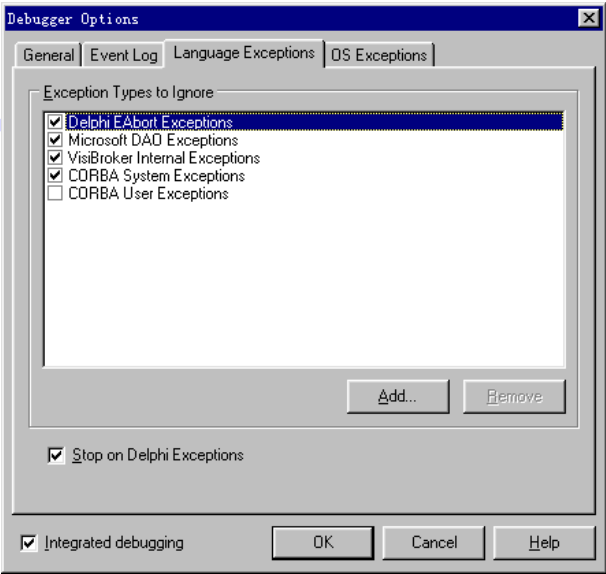


图 12-3 Language Exceptions 标签页

表 12-2 设置调试时可以忽略的异常列表

异常类型	默认设置	对应的异常
Delphi EAbort Exceptions (Delphi Eabort 异常)	忽略	EAbort
Microsoft DAO Exceptions (微软 DAO 异常)	忽略	CDaoException
VisiBroker Internal Exceptions (VisiBroker 内部异常)	忽略	IODictionary<IOUniqueId,dpIOHandler*>::OBJECT_NOT_EXIST
CORBA System Exceptions (CORBA 系统异常)	忽略	CORBA_SystemException 和 CORBA_SystemException*
CORBA User Exceptions (CORBA 用户异常)	不忽略	CORBA_UserException 和 CORBA_UserException *

- Stop on Delphi Exceptions (Delphi 遇到异常时停止执行程序) 如果在调试阶段，希望应用程序提交了一个异常时就终止执行，则应该选中 “Stop On Delphi



Exceptions” 复选框，这是默认设置值。此时，只有在调试阶段遇到可以忽略的异常时，应用程序才不会终止运行；而没有被忽略的 Delphi 异常都会终止程序的继续运行。

- **Integrated debugging (综合调试)** 选择该选项可以激活 Integrated Debugger (综合调试器)，默认情况下选中该复选框。
- **Add 按钮** 通过该按钮可以在忽略的异常列表中添加新的异常类型。单击该按钮后，会打开一个对话框，可以在该对话框的编辑框中输入希望忽略的异常的名字，比如输入数学运算引起的异常 EmathError，此时对话框的显示如图 12-4 所示，单击 OK 按钮确认后，该异常就会添加到忽略异常列表中，同时会自动选择该选项，表示在调试应用程序时，可以忽略该异常及其子异常。

按照相似的方法，可以向列表中添加多个可以被忽略的异常。

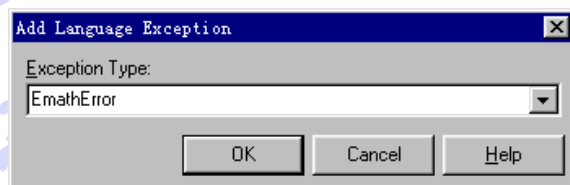


图 12-4 添加异常的对话框

- **Remove 按钮** 该按钮可以从列表中删除一个异常类型。删除方法是先在列表中选择要删除的异常，再单击该按钮，则该异常就被从列表中删除了。

## 2. OS Exceptions 标签页

该标签页用于设置应用程序调试阶段如何处理一些操作系统异常，其显示如图 12-5 所示。该标签页的列表框中列出了所有可以设置的异常。要改变一个异常默认的设置，可以在 Handled By 和 On Resume 两部分选择合适的选项，同时可以通过 Add 或 Remove 按钮添加或删除异常的范围。

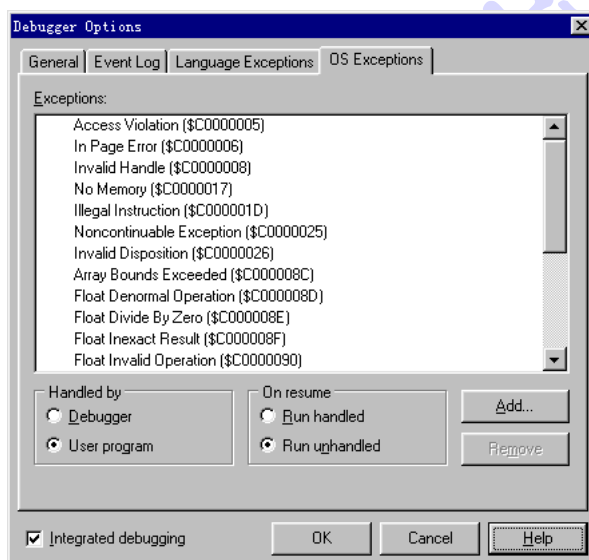


图 12-5 OS Exceptions 标签页

该标签页各个选项的意义如下：

- **Handled By** 指定调试器或应用程序是否处理某个异常。如果在应用程序中已经添加了处理某种异常的代码，则可以选择 **User program**，该选项是默认设置；否则，应选择 **Debugger** 选项，被选中的异常的左边会显示一个红色的圆点。
- **On Resume** 指定在最终交付用户使用的产品中是否继续处理某个异常，或者是否应用程序在不处理某个异常的情况下继续运行。默认选项是 **run unhandled**，即 1 表示不处理该异常。
- **Add 按钮** 该按钮会打开一个异常范围设置对话框，如图 12-6 所示，用于添加用户定义的异常，以便让调试器进行处理。

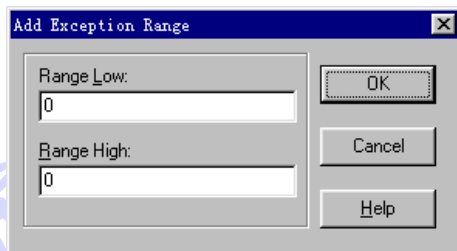


图 12-6 异常范围设置对话框

如果在该对话框中设置了一个最低值和最高值，则任何异常的值出现在该范围内时，应用程序都会停止运行，如果希望在某个具体值上停止程序的运行，则可以将两个选项设置成相同的值。

新添加的异常的范围显示在异常列表框的底部，并且默认用于调试器中。

- **Remove** 从列表中删除当前选择的异常。目前，只能删除使用 **Add** 按钮添加的异常，而不能删除默认的一些异常。

## 第 13 章 建立跨平台应用程序

可以使用 Delphi 6 开发运行在 Windows 和 Linux 操作系统上的跨平台的 32-bit 应用程序。它与 Linux 环境下的 Kylix 相对应, Kylix 实际上是 Inprise 公司在 Linux 操作系统上的 Borland Delphi, 它用于在 Linux 上编译及开发应用程序。如果要开发可以同时运行在 Linux 和 Windows 上的应用程序, 则可以使用 Kylix, 也可以使用 Delphi。这两种应用程序的开发都需要使用共同的 CLX 可视化组件, 但是使用它开发的跨平台应用程序必须在最终的运行平台上编译, 但可以在另一个平台上开发。

### 13.1 建立跨平台应用程序的方法

建立跨平台应用程序与建立一般 Delphi 应用程序的方法一样, 但是需要使用 CLX 可视化组件。要使开发的应用程序是完全跨平台的, 则不要使用操作系统特有的 API (Application Programming Interface, 应用编程接口)。

1. 使用 Delphi 6 建立跨平台应用程序 具体方法如下:

(1) 在 IDE 编辑环境中, 选择 File|New|CLX application 菜单。此时, 组件面板会相应地转换为用于 CLX 应用程序的组件。

(2) 现在可以按照开发 Delphi 应用程序的方法, 但是要记住, 开发的应用程序只能使用 CLX 组件。

(3) 在最终运行的平台上编译并测试该应用程序, 然后对出现的错误进行必要的修改。

**注意:** 当将应用程序移植到 Kylix 环境下时, 需要重新设置项目选项 (project options), 这是因为在 Delphi 中保存项目选项的 .dof 文件在 Kylix 中会重新生成成为 .kof 文件, 并使用默认的设置。如果要保存应用程序的许多编译选项, 可以按 Ctrl+O+O, 该选项放置当前打开文件的开头。

在跨平台的应用程序中, 表单文件的扩展名也由以前的 dfm 修改为 xfm, 这是为了区分使用 CLX 组件开发的表单和使用 VCL 组件建立的表单。xfm 格式的表单既可以在 Windows 下运行, 也可以在 Linux 下运行; 但是 dfm 格式的表单只能运行在 Windows 操作系统下。

2. 使用 Kylix 开发跨平台应用程序 也可以在 Linux 下使用 Kylix 替代 Delphi 来开发跨平台应用程序, 其开发步骤如下:

(1) 在 Linux 下使用 Kylix 开发、编译及测试应用程序。

(2) 将应用程序的源文件移植到 Windows 环境下的 Delphi 中。

(3) 重新使用 “Project/Options” 菜单设置项目选项。

(4) 在 Windows 操作系统上使用 Delphi 重新编译该应用程序。

这样在 Linux 环境下开发的应用程序就可以运行在 Windows 上。

从以上讨论可以看出, 既可以使用 Delphi 6, 也可以使用 Kylix 开发跨平台应用程序。

## 13.2 将 VCL 应用程序移植到 CLX

以前在 Windows 环境下使用 Delphi 开发的应用程序，是否能够移植到 Linux 下呢？答案是肯定的。但是移植的难易程度依赖于该应用程序的复杂性及它们使用了多少与 Windows 有关的组件等内容。

将应用程序从一个平台移植到另一个平台通常使用的方法主要有以下几种：

- **特定平台的移植** 解决的目标是操作系统和底层的 API。这种方式是一项费时、费力的工作，并且只会产生单一的结果。它们建立不同的代码基础，从而使得这些应用程序难以维护。但是，每个移植是针对特定的操作系统设计的，所以可以利用该操作系统的一些优点，因此这种移植的应用程序的运行速度比较快。
- **跨平台的移植** 跨平台的移植是一种快速的开发技术，移植的应用程序可以运行在多个平台上。事实上，移植跨平台应用程序的工作量依赖于已经存在的代码，如果设计的代码没有考虑与平台的无关性，则会遇到与平台有关和与平台无关混合编程的情形。跨平台的开发方法可以快速开发出符合商业要求的应用程序。在开发这种应用程序时，应该分离出跨平台的部分，然后在顶层实现特定的服务。大量基于共享的代码和增强的应用程序体系结构减少了维护程序的费用。
- **模仿 Windows 代码** 这种方式保留代码并移植其使用的 API。这是一种最复杂，也是最昂贵的开发方法，但是使用该方法移植到 Linux 下的应用程序从外观上与 Windows 下的原应用程序非常相似，这种方法在 Linux 上使用了实现 Windows 的一些功能。这种方法开发的代码也非常难于维护，因为它是通过使用 `$IFDEF` 和 `$ENDIF` 语句来区分哪些代码运行在 Windows 下，哪些代码运行在 Linux 下，即相当于在一个应用程序中提供了两套代码。

### 1. 模仿 Windows 代码的移植方法

下面重点说明一下该方法的实现步骤。

(1) 在 Delphi 中打开包含需要改变的应用程序的工程文件。

(2) 将所有的使用 .dfm 扩展名的表单文件拷贝成 .xfm 表单文件，比如，将 `unit1.dfm` 修改为 `unit1.xfm`，同时将单元文件中的 `{SR *.dfm}` 语句修改为 `{SR *.xfm}`，因为 .xfm 格式的表单可以同时运行在 Windows 和 Linux 下。

(3) 修改所有 `uses` 语句，以便在 CLX 中引用正确的单元文件。例如，下面是在 Windows 应用程序中的 `uses` 语句：

```
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;
```

移植为 CLX 应用程序时，可以修改为：

```
uses Windows, Messages, SysUtils, Variants, Classes, QForms, QControls,
QStdCtrls;
```

可见一些单元文件的名字一样，而另一些或者名字进行了改变，或者进行了合并。

(4) 保存拷贝的项目文件，然后在最终运行的平台上重新打开它。这时组件面板会显示可以应用到 CLX 应用程序中的组件。

**注意：**一些 Windows 操作系统中，只可以在其 CLX 应用程序中使用一些不可见 VCL

组件,但是这些组件提供的功能只能运行在 Windows CLX 应用程序中。如果需要将这些应用程序移植到 Linux 下,不要使用不可见的 VCL 组件,或者使用 \$IFDEF 来标注这些只能用于 Windows 下的代码。在同一个应用程序中,不能同时使用 VisualCLX 和 VCL 的可见部分。

(5) 重新编写任何不依赖 Windows 的代码,使这些代码具有与平台无关的特点。也不要使用运行时库中的程序和常量。

(6) 找出在 Linux 中如何实现等价功能的一些不同特点,并使用少量的 \$IFDEF 语句来区分 Windows 的特定信息。下面的例子就是使用 \$IFDEF 语句来区分用于特定平台的代码:

```
[IFDEF MSWINDOWS]
IniFile.LoadFromFile ('c:\test.txt');
[ENDIF]

[IFDEF LINUX]
IniFile.LoadFromFile ('/home/name/test.txt');
[ENDIF]
```

(7) 在所有项目文件中查找路径名的参考。

- 改变 Linux 文件的位置 Linux 中的路径名使用向前的斜线作为分隔符(例如 /usr/lib),文件可以位于不同的目录中。可以使用 SysUtils 单元中定义的 PathDelim 常量来指定该系统使用的路径分隔符。
- 改变驱动器字母 查找驱动器字母的代码,驱动器字母后面一般使用一个冒号“:”。可以使用 SysUtils 单元中定义的 DriveDelim 常量来指定该系统使用的驱动器字母。
- 修改多路径间隔符 如果使用了多个路径,则需要将路径分隔符由分号“;”修改为冒号“:”,可以使用 SysUtils 单元中定义的 DriveDelim 常量来指定该系统使用的路径分隔符。
- 改变文件名 因为 Linux 下文件名是大小写敏感的,应确保你的应用程序不要改变文件名的大小写

(8) 编译并调试你的应用程序。

## 2. 将应用程序移植到 Linux

(1) 将 Windows 下使用 Delphi 开发的应用程序源文件和项目相关的其他文件拷贝到 Linux 计算机上。如果希望应用程序同时运行在 Windows 和 Linux 下,则可以共享源文件,或者使用诸如 FTP 等工具在 ASCII 模式下将文件传输到 Linux 下。

源文件包括单元文件(.pas)、项目文件(.dpr)和包文件(.dpk),与项目相关的文件包括表单文件(.xfm)、资源文件(.res)和工程选项文件(Windows 下的.dof 文件修改为 Kylix 下的.kof 文件)。如果不使用 IDE 环境而使用命令行编译应用程序,则还需要配置文件(Windows 下的.cfg 文件修改为 Kylix 下的.conf 文件)。

(2) 在 Kylix 中打开该项目文件,此时会显示正在使用的 Windows 特点的公告。

(3) 使用 Kylix 编译该项目,通过检查操作信息来查看需要作出哪些进一步的改变。

### 13.3 CLX 与 VCL 的对比

Kylix 使用跨平台的 Borland 组件库来代替 Delphi 中使用的可见组件库 VCL。在 VCL 中, 通过许多控件都可以很容易地访问 Windows 控件; 同样, CLX 可以访问 Qt 共享库中的 Qt 窗口部件 (widget)。Delphi 6 中同时包括了 CLX 和 VCL。

CLX 与 VCL 在外观上非常相似, 并且大部分的组件使用相同的名字, 许多属性的名字也相同。另外, CLX 和 VCL 都可用于 Windows 操作系统中。

CLX 可以划分为以下几部分:

- VisualCLX 该部分包括了本地跨平台的 GUI (图形用户界面) 组件和图形, 该部分组件在 Linux 和 Windows 是不同的。
- DataCLX 客户端数据访问组件, 该部分组件是本地基于客户数据集的多层 client/server 结构的一个子集, 该部分组件在 Linux 和 Windows 下是相同的。
- NetCLX 该部分是 Internet 组件, 包括 Apache DSO 和 GI Web Broker, 该部分组件在 Linux 和 Windows 下是相同的。
- RTL 包括了 Classes.pas 的运行时库, 该部分代码在 Linux 和 Windows 下是相同的。

VisualCLX 下的窗口部件代替了 Windows 的控件。CLX 中的 TWidgetControl 代替了 VCL 中的 TWinControl, 其他组件 (比如 TScrollingWidget) 也具有与上面类似的对应名字, 但是在移植应用程序时, 不需要将 TWinControl 修改为 TWidgetControl, 因为如下面语句的类型声明已经保存在 QControls.pas 源文件中, 以便简化源代码的共享:

```
TWinControl = TWidgetControl;
```

TWidgetControl 控件及其子孙都具有一个 Handle 属性和 Hooks 属性。Handle 属性是对 Qt 对象的引用; 而 Hooks 属性则引用了 hook 对象, 用于处理一些事件。

CLX 中一些单元的名字和类的位置是不同的, 所以在进行移植时, 需要对单元文件或项目文件代码中的 uses 语句进行修改, 删除那些在 CLX 中不存在的单元, 并修改一些单元的名字使之与 CLX 中的单元一致。

1. CLX 与 VCL 的不同之处 尽管为了保持一致性, 大部分 CLX 与 VCL 非常相似, 但是有些特点的的实现还是有所区别, 下面说明二者之间的区别, 以便在建立跨平台应用程序时能够对这些内容引起重视。

(1) 外观上的不同 Linux 下的可视化环境与 Windows 下的环境外观上有所不同。在 Linux 下, 根据用户使用的窗口管理器的不同 (比如使用 KDE 或 Gnome), 一些对话框的显示也不完全相同。

(2) 样式 (style) 除了 OwnerDraw 属性外, Linux 可以使用大部分的应用程序样式, 可以使用 TApplication.Style 属性来设定应用程序图形元素的外观。使用样式时, 窗口部件或应用程序会有一个全新的外观。在 Linux 中仍可以使用属主 (owner) 绘制图形, 但是建议最好使用样式。

(3) 变量 Windows 的 System 单元中所有的变量代码, 在 Linux 中都放在两个新单元 Variants.pas 和 VarUtils.pas 中。而操作系统依赖的代码在 VarUtils.pas 中被单独放置, 它也包含了 Variants.pas 需要的各种内容的通用版本。如果要包括了 Windows 调用的应用



程序转换成一个 CLX 应用程序，则需要将这些调用修改为对 VarUtils.pas 的调用。

如果需要使用变量，则在 uses 语句中必须包括 Variants 单元。如果要清除一个变量，则在 Linux 下需要使用 VarIsClear 函数。

(4) 自定义变量数据处理器 可以为变量定制数据类型，当给变量分配类型时，这会引入操作符的重载。要建立新的数据类型，可以从类 TcustomVariantType 中继承，也可以实例化新的变量类型。

例如，在 VarCmplx.pas 中通过定制变量实现了复杂的机制，它下面的变量操作符：+、-、×、/（不是整除）和负数，也支持下列互逆转换：SmallInt、Integer、Single、Double、Currency、Date、Boolean、Byte、OleStr 和 String 类型。注意，float（浮点）/ordinal（序数）的转换将丢失一些精度。

(5) 注册表 Linux 不能使用注册表来保存配置信息，而是使用文本配置文件和环境变量来保存配置信息，Linux 下的系统配置文件一般放置在 /etc 目录下，例如 /etc/temp，其他的用户配置文件使用隐含文件（前面使用一个小点“.”），例如 .bashrc，它用于保存 bash shell 的设置或 Xdefaults，后者用于设置 X 程序的默认值。

依赖于注册表的代码可以通过使用一个本地的文本配置文件作出改变，例如，可以将其保存到应用程序的目录中。编写一个单元包含所有的注册表功能，但是要将所有输出保存到一个本地的配置文件中，从而可以解决以前对注册表的依赖。

在 Linux 下要将所有信息放置到一个全局文件夹中时，可以将一个全局的配置文件保存到根目录下，这使得所有的应用程序可以访问相同的配置文件。但是，你必须确保已经正确建立了文件的权限及访问的权利。

在跨平台应用程序中也可以 ini 文件，但是，在 CLX 中需要使用 TmemIniFile 代替 TregIniFile。

(6) 其他的区别 CLX 中还有其他的特点会影响应用程序的运行。

1) 在 Kylix 中不能像在 Delphi 中一样通过按 Enter 键来激活一个 click（单击）事件，不能通过按 Enter 键使 ToggleButton 获得切换。

2) TcolorDialog 没有一个 TColorDialog.Options 属性需要设置，因此，不能定制颜色选择对话框的外观和功能。TcolorDialog 也不是一个模型，在 Kylix 中可以使用一个模型对话框来操纵应用程序的标题条，也就是说，你可以选择颜色对话框的父表单，并且在打开颜色对话框时可以进行最小化等处理。

3) 在运行时，Kylix 中的下拉框与 Delphi 中的下拉框是不同的。在 Kylix 中，可以在下拉框的编辑区域输入文本并按 Enter 键来添加一个新的项，如果将 InsertMode 设置为 ciNone，则会取消该功能。也可以向下拉框中添加一个空项（没有字符串）。同时，如果持续按住向下的箭头，也不会停留在列表框的最后一个项上，它会重新循环到顶部的项。

4) TcustomEdit 不能实现 Undo、ClearUndo 和 CanUndo 等功能，所以程序无法实现 undo（取消操作）功能，但用户在应用程序运行时，可以通过在一个编辑框（TEdit）上单击鼠标右键并选择 Undo 命令，来取消他们在一个编辑框中进行的操作。

5) 在 Windows 中，在 OnKeyDown 和 KeyUp 事件中的 Enter 值是 13；在 Linux 中，该值是 4100。如果要检查一个键的核心代码值，比如要检查 Enter 键的值 13，则将 Delphi 应用程序移植到 Kylix 中时，需要修改它。



2. 不能移植的特点 当使用 CLX 代替 VCL 时,许多对象是一样的。但是,这些对象的一些特点会丢失,比如一些属性、方法和事件等。CLX 中缺少如下几个特点:

- 自右向左文件的输出或输入的双向属性(比如 BidiMode)。
- 一般控件上的 bevel(导角)属性(个别对象仍保留该属性)。
- Docking(停放)属性和方法。
- 向后兼容的特点,比如 Win3.1 组件面板和 Ctl3D 提供的组件。
- DragCursor(拖拉光标)和 DragKind(拖拉类型)。

Delphi 支持的一些 Windows 特有的特点不能直接移植到 Linux 环境下,诸如 COM、ActiveX、OLE、BDE 和 ADO 等特点依赖于 Windows 环境,在 Kylix 下是不可用的。两个平台中不同的一些特点见表 13-1,该表同时列出了 Kylix 中等价的特点。

表 13-1 Delphi 和 Kylix 中不同或等同的特点

Delphi/Windows 特点	Kylix/Linux 对应特点
ADO 组件	一般数据库组件
自动控制的服务器	没有对应功能
BDE	DbExpress 和一般的数据库组件
COM+组件(包括 ActiveX)	没有对应功能
DataSnap	没有对应功能
FastNet	没有对应功能
Internet Express	没有对应功能
向前兼容组件(比如 Win 3.1 组件面板上的一些项)	没有对应功能
消息应用程序接口(Messaging Application Programming Interface,MAPI),包括实现 Windows 消息功能的一个标准库	SMTP/POP3 允许你发送、接收及保存电子邮件信息
快速报表	没有对应功能
Web Services(SOAP)	没有对应功能
WebSnap	没有对应功能
Windows API 的调用	CLX 方法、调用 Qt 及 libc,或调用其他的库
Windows 消息	Qt 事件
Winsock	BSD socket

Linux 中等价于 Windows DLL 的是共享对象库(.so 文件),这些库中包含了位置独立的代码(position-independent code, PIC)。Kylix 库模型和包使用.so 文件实现。Kylix 中有下列区别:

- 不允许变量在内容中引用绝对地址(使用 absolute 指令)。
- 全局内存的引用和对外部功能的调用对 EBX 注册表是相对的,它必须通过调用来保存。

如果在 Kylix 和 Delphi 中要使用汇编产生正确的代码,则需要关注全局内存的引用和对外部功能的调用。

3. CLX 和 VCL 单元的比较

VCL 和 CLX 中的所有对象都是在单元文件(.pas 源文件)中定义,例如,可以在 System

单元中找到 Tobject 对象的实现，以及 Classes 单元定义了基类 Component。当将一个对象从组件面板拖拉到一个表单中，或在应用程序中使用了一个对象，则定义该对象的单元文件会添加到当前表单对应单元文件的 uses 语句中，这样可以使编译器知道哪一个单元文件需要连接到项目中。

VCL 与 CLX 中都具有的单元比较见表 13-2；CLX 具有而 VCL 没有的组件见表 13-3；而 VCL 具有而 CLX 没有的组件见表 13-4，这些组件由 Windows 的特点决定。

表 13-2 VCL 与 CLX 中都具有的单元比较

VCL	CLX	VCL	CLX	VCL	CLX
ActnList	QActnList	DBLogDlg	DBLogDlg	Provider	Provider
Buttons	QButtons	DBXpress	DBXpress	Qt	Qt
CheckLst	QCheckLst	Dialogs	QDialogs	Search	QSearch
Classes	Classes	DSIntf	DSIntf	Sockets	Sockets
Clipbrd	QClipbrd	ExtCtrls	QExtCtrls	StdActns	QStdActns
ComCtrls	QComCtrls	FMTBCD	FMTBCD	StdCtrls	QStdCtrls
Consts	Consts、Qconsts 和 RTLConsts	Forms	QForms	SqlConst	SqlConst
Contrns	Contrns	Graphics	QGraphics	SqlExpr	SqlExpr
Controls	QControls	Grids	QGrids	SqlTimSt	SqlTimSt
DateUtils	DateUtils	HelpIntfs	HelpIntfs	SyncObjs	SyncObjs
DB	DB	ImgList	QImgList	SysConst	SysConst
DBActns	QDBActns	IniFiles	IniFiles	SysInit	SysInit
DBClient	DBClient	Mask	QMask	System	System
DBCommon	DBCommon	MaskUtils	MaskUtils	SysUtils	SysUtils
DBConnAdmin	DBConnAdmin	Masks	Masks	Types	Types 和 QTypes
DBConsts	DBConsts	Math	Math	TypInfo	TypInfo
DBCtrls	QDBCtrls	Menus	QMenus	Variants	Variants
DBGrids	QDBGrids	Midas	Midas	VarUtils	VarUtils
DBLocal	DBLocal	MidConst	MidConst	Provider	Provider
DBLocalS	DBLocalS	Printers	QPrinters	Qt	Qt

表 13-3 CLX 特有的组件

单元	描述
DirSel	用于目录选择
QStyle	决定 GUI 外观和感觉

表 13-4 VCL 特有而 CLX 没有的组件

VCL	CLX	VCL	CLX	VCL	CLX
ADOConst	没有 ADO 特点	DBTables	没有 BDE 特点	OleCtrls	没有 ADO 特点
ADODB	没有 ADO 特点	DdeMan	没有 DDE 特点	OLEDDB	没有 ADO 特点
AppEvnts	没有 TApplicationEvent 对象	DRTable	没有 BDE 特点	OleServer	没有 ADO 特点
AxCtrls	没有 ADO 特点	ExtActns	Delphi 6 的新特点	Outline	过时的
BdeConst	没有 BDE 特点	ExtDlgs	没有图像对话框	Registry	Windows 特有的对注册表的支持
ComStrs	没有 COM 特点	FileCtrl	过时的	ScktCnst	被 Sockets 代替
ConvUtils	Delphi 6 的新特点	ListActns	Delphi 6 的新特点	ScktComp	被 Sockets 代替
CorbaCon	没有 Corba 特点	MConnect	没有 COM 特点	SConnect	不支持连接协议
CorbaStd	没有 Corba 特点	Messages	Windows 特有的领域	StdConvs	Delphi 6 的新特点
CorbaVCL	没有 Corba 特点	MidasCon	过时的	SvcMgr	对 NT 服务的支持
CtlPanel	没有对 Windows 控件面板的支持	MPlayer	Windows 特有的媒体播放器	Tabnotbk	过时的
DataBkr	可能以后出现在 upsell 中	Mtsobj	没有 COM 特点	Tabs	过时的
DBCGrids	没有 BDE 特点	MtsRdm	没有 COM 特点	ToolWin	没有 dock 特点
DBExcept	没有 BDE 特点	Mtx	没有 COM 特点	VarCmplx	Delphi 6 的新特点
DBInpReq	没有 BDE 特点	mxConsts	没有 COM 特点	VarConv	Delphi 6 的新特点
DBLookup	过时的	ObjBrkr	可能以后出现在 upsell 中	VCLCom	没有 COM 特点
DbOleCtl	没有 COM 特点	OleConstMay	没有 COM 特点	WebConst	Windows 特有的 constants
DBPWDlg	没有 BDE 特点	OleCtnrs	没有 COM 特点	Windows	Windows 特有的 (API)

#### 4. 不能移植的一些特点

当建立一个 CLX 对象时,可以通过将该对象放置到表单设计器中来隐含该对象,也可以通过使用该对象的 Create 方法使之显式出现在代码中,与 widget 关联的一个实例也会被建立。该 CLX 对象拥有这个 widget 实例,当调用 Free 方法删除该 CLX 对象,或通过 CLX

对象的父容器自动删除该对象时，该 widget 实例也会被删除。这与 VCL 在 Windows 应用程序中实现的功能相同。

当在代码中显式建立一个 CLX 对象时，可以调用 Qt 接口库，比如 `QWidget_Create()`，这样会建立一个不属于 CLX 对象的 Qt widget 实例，并在其建立期间向该 CLX 对象传递一个 Qt widget 实例。该 CLX 对象没有传递给它的 Qt widget 对象，因此，当使用这种方式在建立了该对象后调用 `Free` 方法时，只会销毁该 CLX 对象，而不会销毁下面的 Qt widget 实例。这与 VCL 中是不同的。

通过使用 `OwnHandle` 方法，一些 CLX 对象让你假设拥有下面的 widget。在调用了 `OwnHandle` 方法后，如果删除了 CLX 对象，则也会销毁下面的 widget。

5. Windows 和 Linux 共享源代码文件 如果你希望开发的应用程序同时运行在 Windows 和 Linux 上，则可以通过共享源文件，使得两个操作系统都能访问它。可以通过多种方法实现它，比如将源文件放置在服务器上，或在 Linux 上使用 Samba，并通过 Linux 和 Windows 计算机上微软网络的共享来访问这些文件，也可以将源文件放置在 Linux 上，并在 Linux 上建立一个共享驱动器，或者在 Windows 上保存源文件，并在 Windows 计算机上建立一个共享目录，以便 Linux 计算机可以访问。

可以使用同时支持 VCL 和 CLX 的对象，在 Kylix 上继续开发这些应用程序，然后可以在 Linux 和 Windows 计算机分别编译这些文件。

Delphi 中的表单文件 `.dfm`，在 Kylix 中被修改为 `.xfm` 文件。如果在 Delphi 或 Kylix 中建立一个 CLX 应用程序，则 `.xfm` 文件会代替 `.dfm` 文件。如果需要编写跨平台的应用程序，则 `.xfm` 文件可以同时用于 Delphi 和 Kylix 中。

6. Windows 和 Linux 环境的差别 通常，跨平台意味着一个应用程序可以不经任何修改就可以同时运行在 Windows 和 Linux 操作系统上，两个操作系统的具体差别见表 13-5。

表 13-5 Windows 和 Linux 环境的差别

区别	说明
文件名大小写敏感	在 Linux 中，大写字母与小写字母是不同的，即文件 <code>Test.txt</code> 与 <code>test.txt</code> 是两个不同的文件
一行的结束字符	在 Windows 中，文本行以回车/换行符 (CR/LF，即 ASCII 13 + ASCII 10) 结束，但是 Linux 的行结束符是 LF。Kylix 中的代码编辑器可以处理这种不同，在从 Windows 向 Linux 引入代码时应注意这一点
文件的结束字符	在 DOS 和 Windows 中，文本文件的结束字符是 #26 (Ctrl+Z)，即使该字符后还有数据；Linux 中没有特定的文件结束符，文本数据结束的位置就是文件的结尾
批处理文件/shell 脚本	Linux 中与 Windows 中的 <code>.bat</code> 文件等同的是 shell 脚本。一个脚本是一个文本文件，其中包含了指令、保存功能，并通过 <code>chmod +x&lt;脚本文件&gt;</code> 的命令来执行该文件，要运行该文件，只需要输入其名字。脚本语言则依赖于 Linux 中使用的 shell，通常使用 <code>bash</code>
命令确认	在 DOS 或 Windows 中，如果试图删除一个文件或文件夹，则会弹出一个对话框要求确认。但是，在 Linux 中一般不会询问而直接删除。在 Linux 中没有办法可以恢复一个已经进行的删除，除非使用其他媒体备份了这些文件

(续表)

区别	说明
命令反馈	如果一个命令在 Linux 中成功执行, 则它会重新显示该命令的提示符, 而没有状态信息
命令开关	Linux 使用一个中划线来表示一个命令开关或使用两个中划线来表示多个字符选项, 而 DOS 中使用正斜线或中划线来表示
配置文件	<p>在 Windows 中, 配置信息放置在注册表中或一些文件中, 比如 autoexec.bat; 而在 Linux 中, 配置文件以隐含文件的形式建立并使用一个小点开头, 许多配置文件都放置在 /etc 目录或系统的主目录中。</p> <p>Linux 也使用环境变量, 比如 LD_LIBRARY_PATH, 它用于查找库的路径, 其他重要的环境变量有:</p> <p>HOME 系统主目录 (/home/sam)</p> <p>TERM 终端的类型 (xterm, vt100, console)</p> <p>SHELL 系统 shell 的路径 (/bin/bash)</p> <p>USER 用户注册名字 (sfuller)</p> <p>PATH 对程序的搜索列表</p> <p>它们都在 shell 或在 rc 文件 (比如 .bashrc) 中指定</p>
DLL	在 Linux 中可以使用共享对象文件 (.so); 而在 Windows 中用动态链接库 (DLLs)
异常	在 Linux 中, 操作系统异常通过发信号调用
可执行文件	在 Linux 中, 可执行文件不需要扩展名; 在 Windows 中, 可执行文件使用 .exe 为扩展名
文件名扩展符	Linux 不使用文件扩展名来表示文件的类型或与应用程序相关的文件
Linux 中的文件权限	<p>在 Linux 中, 文件或目录对文件的属主、所在的组及其他人分配了可读、可写及运行的权限, 例如:</p> <p>-rwxr-xr-x</p> <p>自左向右各个符号的意义如下: - 是文件的类型, (- = 普通文件, d = 目录, l = 链接); rwx 是文件拥有者的权限 (read、write、execute); r-x 是文件属主所在组的权限 (read、execute); -x 是其他用户的权限 (read、execute)。根用户 (超级用户) 可以覆盖这些权限。</p> <p>你需要确信应用程序运行在正确的用户下, 并且对需要的文件有适当的访问权限</p>
Make 工具	Borland 的 make 工具不能用于 Linux 平台, 但是可以使用 Linux 的 GNU make 工具
多任务	Linux 完全支持多任务, 你可以同时运行几个程序 (在 Linux 中成为进程)。也可以通过在命令后使用 & 将进程放在后台运行, 并继续运行其他的任务, Linux 也允许使用几个会话
路径名	Linux 使用正斜线, 而 DOS 使用反斜线。一个运行库中定义的 PathDelim 常量可用于指定该平台中使用的合适字符
搜索路径	当运行程序时, Windows 总是首先检查当前的目录, 然后查找 PATH 环境变量; Linux 从来不会查找当前目录, 但是会查找 PATH 中列出的目录。要运行当前目录下的程序, 通常可以在它前面输入 ./, 也可以修改 PATH 来包括 ./, 并将其作为查找的第一个路径
搜索路径分隔符	Windows 使用分号作为搜索路径的分隔符; 而 Linux 使用冒号。可以通过运行库中的 PathDelim 常量来定义分隔符

(续表)

区别	说明
符号链接	Linux 中的符号链接是一个指向磁盘上另一个文件的特殊文件, 将符号链接放置在全局的 bin 目录中来指向应用程序的主文件, 并且不能修改系统的搜索路径, 要通过 ln (link) 命令来建立一个符号链接。 Windows 有 GUI 桌面的快捷键, 为了使一个程序可用于命令行中, Windows 安装程序会修改系统的搜索路径

7. Linux 的目录结构 Linux 与 Windows 有完全不同的目录结构, 任何文件和设备的驱动程序, 可以安装到文件系统的任何位置。Linux 中通常使用的一些目录见表 13-6。

注意: Linux 的目录使用正斜线, 而 Windows 使用反斜线。开始的斜线表示 Linux 的根目录。

表 13-6 Linux 常用的目录

目录	说明
/	整个 Linux 文件系统最顶层目录的根
/root	根文件系统, 超级用户的主目录
/bin	命令、工具
/sbin	系统工具
/dev	以文件形式显示的设备
/lib	库
/home/username	用户拥有的文件, username 是用户的注册名
/opt	选项
/boot	系统启动时调用的内核
/etc	配置文件
/usr	应用程序、一般程序, 通常包括一些目录, 比如/usr/spool、/usr/man、/usr/include、/usr/local
/mnt	系统上安装的其他媒体, 比如一个 CD 或一个软盘驱动器
/var	日志、消息及 spool 文件
/proc	虚拟文件系统及报告系统的统计信息
/tmp	临时文件

注意: 不同的 Linux 有时会将文件放置到不同的目录中, 在 Red Hat Linux 中可能将一个工具放置到/bin 目录下, 而 Debian Linux 可能将其放置到/usr/local/bin 目录下。

另外, 可以到 [www.pathname.com](http://www.pathname.com) 站点了解 UNIX/Linux 分级文件系统更多的细节, 并阅读 Filesystem Hierarchy Standard (文件系统分级标准) 文档。

## 13.4 编写可移植的代码

如果要编写跨平台应用程序, 可以编写在不同条件下编译的代码。如果使用条件编译器, 在维持 Windows 代码的同时, 可以增加适合 Linux 操作系统的代码。

要建立可以很容易地在 Windows 和 Linux 平台上移植的应用程序, 应记住以下几点:



- 减少或隔离对特定平台的 API (Win32 或 Linux) 的调用, 而使用 CLX 方法替代。
- 取消在一个应用程序内 Windows 消息的构造 (PostMessage、SendMessage)。
- 使用 TmemIniFile 代替 TregIniFile。
- 注意观察并保持文件和目录名的大小写。
- 删除任何外部汇编器 TASM 代码, 因为 GNU 汇编器不支持 TASM 语法。
- 尽量使用与平台无关的运行库程序并使用 System、SysUtils 和其他运行库单元中的常量。例如, 使用 PathDelim 常量来隔离在不同平台中使用的代码。

另外, 在两个平台上都使用多字节的字符, Windows 代码的每个多字节字符使用 2 个字节; Linux 中的多字符编码在每个字符中可以有多字节 (按照 UTF-8 标准最多使用 6 个字节)。通过使用 SysUtils 单元中的 StrNextChar 函数, 可以使两个平台相互适应。比如, 下面的 Windows 代码:

```
while p^ <> #0 do
begin
  if p^ in LeadBytes then
    inc (p);
  inc (p);
end;
```

如果使用与平台无关的代码取代它, 则代码如下:

```
while p^ <> #0 do
begin
  if p^ in LeadBytes then
    p := StrNextChar (p)
  else
    inc (p);
end;
```

该例子在平台间是可移植的, 并且支持 2 个字节和多字节, 但是仍要避免调用非多字节过程时对性能的影响。

如果使用运行库函数无法解决问题, 则在你的程序中应与特定平台有关的代码封装到一段程序或一个子程序中。但是应尽量显示 \$IFDEF 块的使用, 以便维持源代码的可读性和可移植性。在 Linux 中没有定义条件符号 WIN32。而已经定义的条件符号 LINUX 则表示源代码是适合 Linux 平台编译的。

1. 使用条件指令 在代码中使用 \$IFDEF 编译器指令是合理的方法, 可以根据条件决定代码是运行在 Windows 还是 Linux 平台上。但是, 因为 \$IFDEF 指令使得源代码更难于理解及维护, 所以应该理解如何合理地使用 \$IFDEF 命令。在使用该指令时, 最主要的问题是应考虑为何需要使用 \$IFDEF 命令? 如果不使用该命令是否也可以解决问题?

在跨平台应用程序中使用 \$IFDEF 命令, 应遵循如下原则:

(1) 尽量不要使用 \$IFDEF。只有代码被编译时才会对 \$IFDEF 包括的源代码进行取舍。与 C/C++ 不同, Delphi 不需要单元源 (头文件) 来编译一个项目。对于大多数 Delphi 项目来说, 很少会重新建立所有的源代码。

(2) 在包文件 (.dpk) 中不要使用 \$IFDEF。应限制它们用于源文件中。当进行跨平台的开发时, 设计组件应建立两个设计时包, 任何一个包都不要使用 \$IFDEF。

(3) 通常情况下, 使用 \$IFDEF MSWINDOWS 语句来测试任何 Windows 平台, 包括

WIN32, 可使用 `$IFDEF WIN32` 来区分特定的 Windows 平台, 比如区分 32-bit 与 64-bit Windows。不要将代码限制为 WIN32, 除非确信将来应用程序不会运行在 64-bit Windows 上。

(4) 除非绝对需要, 避免使用 `$IFNDEF` 等否定的测试指令, 因为 `$IFNDEF LINUX` 不等于 `$IFDEF MSWINDOWS`。

(5) 避免 `$IFNDEF` 和 `$ELSE` 结合, 使用 `$IFDEF` 代替 `$IFNDEF` 可增加代码的可读性。

(6) 避免在平台敏感的 `$IFDEF` 语句中使用 `$ELSE` 语句, 分别为 `LINUX`-和 `MSWINDOWS`-特定代码使用独立的 `$IFDEF` 块, 而不要用 `$IFDEF LINUX/$ELSE` 或 `$IFDEF MSWINDOWS/$ELSE` 替代它。

例如, 以前的代码可能包含如下代码:

```
{IFDEF WIN32}
    (32-bit Windows代码)
{$ELSE}
    (16-bit Windows代码)    //!! 可能是因为疏忽, Linux没有使用该代码
{$ENDIF}
```

对于任何在 `$IFDEF` 中不可移植的代码, 不能编译源代码比使平台进入 `$ELSE` 语句并且在运行时神秘地失败可能更好。编译比运行更容易找到失败的原因。

(7) 对于复杂的测试应使用 `$IF` 语句, 使用 `$IF` 语句中的一个 `boolean` 表达式来替代嵌套的 `$IFDEF` 语句, 此时应使用 `$IFEND` 指令来结束 `$IF` 语句, 而不是使用 `$ENDIF` 指令。这允许在 `$IFDEF` 中放置 `$IF` 表达式以便从以前的编译器中隐藏新的 `$IF` 语句。

2. 终止条件指令 使用 `$IFEND` 指令可终止 `$IF` 和 `$ELSEIF` 条件指令, 这允许从以前使用了 `$IFDEF/$ENDIF` 语句块的编译器中隐藏 `$IF/$IFEND` 语句块。以前的编译器不能识别 `$IFEND` 指令, `$IF` 指令只能使用 `$IFEND` 指令终止, 可以使用 `$ENDIF` 指令来终止以前一些旧格式的指令, 比如 `$IFDEF`、`$IFNDEF`、`$IFOPT` 等。

注意: 当在 `$IFDEF/$ENDIF` 中嵌套一个 `$IF` 指令时, 不要使用与 `$IF` 结合的 `$ELSE`, 以前的编译器会将 `$ELSE` 看作是 `$IFDEF` 块的一部分, 从而产生编译错误。在这种情况下, 可以使用 `{ELSEIF True}` 来替代 `{ELSE}`, 如果首先取用了 `$IF`, 则不会取用 `$ELSEIF`, 因为以前的编译器不知道 `$ELSEIF`。隐藏 `$IF` 用于向后的兼容, 主要是用于那些希望自己的代码可以运行在几个不同的版本中的第三方供应商和应用程序开发者。

`$ELSEIF` 是 `$ELSE` 和 `$IF` 的结合体, `$ELSEIF` 指令用于编写多个条件块, 但是只会选取一个条件块。例如下面的代码:

```
{IFDEF doit}
    do_doit
{$ELSEIF RTLVersion >= 14}
    goforit
{$ELSEIF somestring = 'yes'}
    beep
{$ELSE}
    last chance
{$IFEND}
```

上面的 4 种情况中只会执行一种。如果前面的 3 个条件都不能满足, 则执行 `$ELSE` 语句。 `$ELSEIF` 必须使用 `$IFEND` 结束, `$ELSEIF` 不能显示在 `$ELSE` 之后, 该条件块就像

\$IF...\$ELSE 语句一样从上向下顺序执行。在该例子中如果没有定义 `doit`，则 `RTLVersion` 是 15，并且 `somestring = 'yes'`，则会执行 `goforit` 块，而不会执行 `beep` 块，即使两个条件都能满足。

如果忘记使用 `$ENDIF` 来结束 `$IFDEF` 语句，则编译器在源文件的底部会报告下列信息：

```
Missing ENDIF
```

如果源文件中有多条 `$IF/$IFDEF` 指令，则很难确定是哪一个出现了问题，Kylix 或 Delphi 在 `$ENDIF/$IFEND` 不匹配的最后一个 `$IF/$IFDEF` 编译器指令的源代码行报告下列错误信息：

```
Unterminated conditional directive
```

可以从该位置开始查找错误。

3. 发出消息 `$MESSAGE` 编译指令可以像编译器一样允许源代码发出提示、警告和错误等信息。其语法如下：

```
{ $MESSAGE HINT|WARN|ERROR|FATAL 'text string' }
```

消息的类型是可选的。如果没有说明消息类型，则缺省值是 `HINT`。该语句中必须使用一个文本字符串，并且必须使用单引号包括起来。下面是几个例子：

```
{ $MESSAGE '默认值是HINT!' }           //发出提示
{ $Message Hint '注意该指令的格式!' }   //发出一个提示
{ $Message Warn '请注意检查源代码是否有错误!' } //发出一个警告
{ $Message Error '无法实现!' }           //发出一个错误并继续编译
{ $Message Fatal '出现错误!' }           //发出一个错误并终止编译器
```

4. 包括内嵌编译器代码 如果在 Windows 应用程序中包括了内嵌的编译器代码，则在 Linux 中不能使用同样的代码，因为 Linux 中需要位置独立的代码（`position-independent code, PIC`），Linux 共享对象库（与 DLL 等价）需要所有的代码在不修改的情况下在内存中重定位，这主要会影响使用全局变量或其他绝对地址的内嵌汇编程序，或调用了外部函数的内嵌汇编程序。

对于那些只包含 Object Pascal 代码的单元，编译器在需要时会产生 PIC，PIC 单元文件使用一个 `.dpu` 扩展名（代替 `.dcu`）。最好的方法是将每个 Pascal 单元源文件分别编译为 PIC 和非 PIC 格式，使用 `-p` 编译器开关来产生 PIC。预编译单元文件在两种表单中都可用。

你可能希望根据编译的是一个共享库或是一个可执行文件而使用不同的代码汇编程序，则可以使用 `{ $IFDEF PIC }` 指令来区分汇编代码的这两个版本，也可以使用 Object Pascal 重新编写程序以避免出现问题。

下面是使用内嵌的汇编代码时应遵循的一些 PIC 规则：

（1）PIC 需要相对于 `EBX` 寄存器的所有内存引用，它包含当前模块的基地址指针（在 Linux 中调用全局偏移量表（`Global Offset Table`）或 `GOT`），所以可以将下面代码：

```
MOV EAX,GlobalVar           // GlobalVar表示全局变量
```

修改为：

```
MOV EAX,[EBX].GlobalVar
```

（2）PIC 要求通过调用汇编代码来保存 `EBX` 寄存器（与 WIN32 一样），在调用外部的函数以前则需要恢复 `EBX` 寄存器（与 WIN32 不同）。

（3）当 PIC 两码事运行在基本的执行状态，它会降低性能并产生更多的代码。在共享对象上没有任何选择，但是在可执行文件上则可能获得更高的性能。

5. 消息和系统事件 在 Linux 和 CLX 中, 消息循环和事件的工作方式是不同的, 但是这种区别主要影响组件的编写。多数组件和属性编辑器移植比较容易, TObject.Dispatch 和类的消息方法语法可以很好地运行在 Linux 上, 但是在 Linux 下, 操作系统的通知是使用系统的事件进行处理, 而不是使用消息。

要在跨平台应用程序中建立一个事件处理程序, 则可以覆盖表 13-7 中描述的方法, 并编写自定义的消息来代替对 Windows 消息的响应。在进行覆盖时, 应调用继承方法, 以便任何缺省的进程仍能发生。

表 13-7 Linux 中使用的方法

方法	描述
ChangeBounds	用于 TwidgetControl 调整大小时, 有点类似 Windows 中的 WM_SIZE 或 WM_MOVE 消息。Qt 基于客户区域设置 widget 的几何形状, VCL 使用整个的控件大小, 它包括了 Qt 对框架的引用
ChangeScale	当调整控件大小时会自动调用该方法。用于改变一个表单的比例和它的所有控件以便适应不同的屏幕分辨率或字体大小。因为该方法会修改控件的 Top、Left、Width 和 Height 等属性, 所以它会改变控件及其子孙的位置和大小
ColorChanged	当控件的颜色改变时调用该方法
CursorChanged	当鼠标指针改变形状时调用该方法, 当通过该 widget 时, 鼠标指针呈现该形状。
EnabledChanged	当应用程序改变一个窗口或控件的激活状态时调用该方法
FontChanged	当字体资源的集合改变时调用该方法。它会设置 widget 的字体并通知所有子孙进行的改变, 有点类似 Windows 中的 WM_FONTCHANGE 消息
PaletteChanged	当系统面板改变时调用该方法
ShowHintChanged	当显示或隐藏一个控件的帮助提示时调用该方法
StyleChanged	当窗口或控件的 GUI 样式改变时调用该方法
TabStopChanged	当表单中的制表符导航顺序改变时调用该方法
VisibleChanged	当显示或隐藏一个控件时调用该方法
WidgetDestroyed	当销毁控件下面的 widget 时调用该方法

Qt 是一个 C++ 工具, 所以它的所有组件都是 C++ 对象。CLX 是使用 Object Pascal 编写的, 而 Object Pascal 不是直接继承自 C++ 对象。另外, Qt 在少数位置使用多种继承, 所以 Delphi 包括了一个接口层, 用来将所有的 Qt 类转换为一系列对应的 C 函数, 然后将其打包进 Linux 的共享对象和 Windows 的 DLL 中。

每个 TwidgetControl 对象都有 CreateWidget、InitWidget 和 HookEvents 等虚拟方法, 总是可以对它们重载。CreateWidget 建立一个 Qt 组件, 并向 Fhandle 私有域变量分配 Handle。InitWidget 在构造完组件后被调用, 而 Handle 是无效的。

Delphi CLX 的一些属性分配已经将 Create 构造器修改为 InitWidget, 这可以延迟 Qt 对象的构造, 直到真正需要时才构造它。例如, 如果有一个名字为 Color 的属性, 则在 SetColor 中可以使用 HandleAllocated 检查是否有一个 Qt 句柄, 如果分配了一个句柄, 则可以调用 Qt 来设置其颜色; 如果没有, 则可以将该值保存到一个私有域变量中, 并且在 InitWidget 中可以设置该属性。

Linux 支持两种类型的事件: Widget 和 System。HookEvents 是一种虚拟的方法, 它会

将 CLX 控件事件方法连接到一个特殊的可以与 Qt 对象通信的 hook（钩子）对象中。该钩子对象实际上是一系列的方法指针，Kylix 中的 System 事件要通过 EventHandler，它基本可以代替 Windows 中的 WndProc 函数。

6. 在 Linux 中编程的区别 Linux 的宽字符 wchar\_t 是每个字符有 32 位。Object Pascal 广义字符支持的 16-bit Unicode 是 Linux 和 GNU 库支持的 32-bit UCS 标准的一个子集，Pascal 的宽字符数据在作为 wchar\_t 传递到一个 OS 函数以前，必须扩充到每个字符为 32 位。另外，Linux 中的宽字符串类似于 Windows 的长字符串。

Linux 中的多字节处理是不同的。在 Windows 中，多字节字符（multibyte characters，MBCS）表示 1 个或 2 个字节的字符编码；而在 Linux 中，它们表示 1~6 个字节。

AnsiStrings 可以表示多字节的字符序列，这依赖于用户所在位置的设置。在日本、中国、希伯莱和阿拉伯等国家，Linux 编码的多字节字符与 Windows 在同一位置的编码是不同的。Unicode（统一编码的字符标准）可以移植，但是多字节却不能移植。

在 Linux 中不能使用绝对地址的变量。下面的语句不被 PIC 支持，在 Delphi 中也是不允许的：

```
var X: Integer absolute $34;
```

## 13.5 编写跨平台数据库应用程序

在 Windows 中，Delphi 提供了几种选择来访问数据库，包括使用 ADO、BDE 和 InterBase Express。但是这三种选择在 Kylix 中是不可用的。可以使用一种新的跨平台数据访问技术 dbExpress 来建立数据库应用程序，它也可以用于 Windows 中，这是 Delphi 6 中新添加的功能。

在将以前开发的数据库应用程序移植到 dbExpress 以便能够运行在 Linux 上时，应该搞清楚使用 dbExpress 与使用其他数据访问机制的区别。这些区别出现在不同的级别上：

- 在最低级别上，有一个在应用程序和数据库服务器通信的层，该层可能是 ADO、BDE 或 InterBase 客户软件，该层会被 dbExpress 代替，它包括了一系列方便的驱动程序用于动态的 SQL 语句处理。
- 低级的数据访问被打包到数据模块或表单中的一系列组件中，这些组件包括连接到数据库服务器的数据库连接组件和从服务器提取数据的数据集。尽管由于 dbExpress 游标的单向性特点会有一些重要的区别，但是在该级别上这些区别却很小，这是因为它们继承自一个共同的祖先。
- 在用户界面的级别上差别最小。CLX 数据感知的控件被设计成尽可能地类似于相应的 Windows 控件，用户界面上最主要的区别是为适应缓冲更新而进行的改变。

1. dbExpress 的差异 在 Linux 中，dbExpress 管理与数据库服务器的通信，dbExpress 包含了一系列方便的驱动程序来实现一些通用接口。每个驱动程序都是一个共享对象（.so 文件），必须链接到应用程序中。因为 dbExpress 被设计成跨平台的，所以在 Windows 中，它作为一系列动态链接库（.dlls）也是可用的。

与其他的数据访问层一样，dbExpress 需要数据库供应商提供的客户软件，另外，它使用一种特定的数据库驱动程序，增加了两个配置文件，即 dbxconnections 和 dbxdrivers。这



比 BDE 需要的内容要少很多, 因为 BDE 不但需要 Borland 数据库引擎库 (Idapi32.dll), 而且还需要特定的数据库驱动程序和大量其他的支持库。

下面是 dbExpress 和其他数据访问层的主要区别, 在移植应用程序时应修改这些内容:

- dbExpress 使用一种到远程数据库的简单而快速的路径。这样会由于简单而直接的数据访问来大大提高应用程序的性能。
- dbExpress 可以处理查询和存储过程, 但是不支持打开的基表。
- dbExpress 只返回单方向的游标。
- dbExpress 没有对执行 INSERT、DELETE 或 UPDATE 查询能力的内置更新支持。
- dbExpress 没有元数据缓冲, 设计时元数据访问接口是使用核心的数据访问接口显示的。
- dbExpress 只执行用户请求的查询, 由于没有引入任何额外的查询来优化了数据库的访问。
- dbExpress 在内部管理一个记录缓冲区或一块记录缓冲区, 这与 BDE 是不同的, 因为 BDE 中要求客户要分配内存以用于缓存记录。
- dbExpress 不支持不是基于 SQL 的本地基表, 比如 Paradox、dBase 或 FoxPro 等。
- dbExpress 中提供了 InterBase、Oracle、DB2 和 MySQL 的驱动程序。使用其他的数据库服务器时, 则不但要将数据移植到这些数据库中, 而且要为该数据库服务器编写一个 dbExpress 的驱动程序, 或者获得一个第三方的用于该数据库服务器的 dbExpress 驱动程序。

2. 组件级别的差别 当编写 dbExpress 应用程序时, 需要使用一些与以前的数据库应用程序中使用的不同的数据访问组件。dbExpress 组件与其他数据访问组件共享相同的基类, 即 TDataSet 和 TCustomConnection, 这意味着 dbExpress 组件与以前应用程序中使用组件的许多属性、方法和事件是相同的。

在 Windows 环境下经常使用的 InterBase Express、BDE 和 ADO 数据库组件与在 Linux 和跨平台的应用程序中经常使用的 dbExpress 组件的对应关系, 见表 13-8。

表 13-8 dbExpress 组件与其他组件的比较

InterBase Express 组件	BDE 组件	ADO 组件	dbExpress 组件
TIBDatabase	TDatabase	TADOConnection	TSQLConnection
TIBTable	TTable	TADOTable	TSQLTable
TIBQuery	TQuery	TADOQuery	TSQLQuery
TIBStoredProc	TStoredProc	TADOStoredProc	TSQLStoredProc
TIBDataSet		TADODataSet	TSQLDataSet

DbExpress 的数据集比其他类似的数据集有更多的限制, 因为它们不支持编辑, 并且只允许向前导航。由于缺乏编辑和导航功能, 所以大部分 dbExpress 并非直接使用 dbExpress 数据集开发, 而是将 dbExpress 数据集连接到一个客户数据集, 该客户数据集在内存中缓存记录, 并且支持编辑和导航。

注意, 对于非常简单的应用程序, 可以使用 TSQLClientDataSet 代替一个 dbExpress 数据集连接到一个客户数据集, 这样比较简单, 因为在被移植的应用程序和最终移植到的应



用程序中间有一一对应的组件，但是显式地将 dbExpress 数据集连接到一个客户数据集缺乏灵活性。所以对于大部分应用程序，建议使用 dbExpress 连接到一个 TclientDataSet 组件。

3. 用户接口级的区别 由于 CLX 数据控件与 Windows 的对应控件非常相似，所以在移植数据库应用程序时，用户接口部分只需要很少的修改。在用户接口级最主要的区别是 dbExpress 或客户数据集提供的数据库方法不同。

如果在应用程序中只使用 dbExpress 数据集，则必须调整用户界面以符合该数据集不支持编辑及只支持向前数据导航的特点，例如，可能需要删除一些允许用户移动到以前的记录。由于 dbExpress 数据集不能缓存数据，所以不能使用表格的形式 (DBGrid) 来显示数据，因为一次只能显示一条记录。

如果已经将 dbExpress 连接到一个客户数据集，则用户界面中可以编辑及导航的控件应该仍能使用，移植时只需要重新将其连接到该客户数据集。此时主要的考虑是写到数据库的更新。缺省时，Windows 上的大部分数据集在提交时自动将修改的数据写入到数据库服务器中，比如，当用户移动到新的记录时就会进行更新。而客户数据集总是将修改的数据保存在内存中，直到提交一个事务时才会将更新写入到数据库中。

4. 将数据库应用程序移植到 Linux 将数据库应用程序移植到 dbExpress 时，允许建立可以运行在 Windows 和 Linux 上的跨平台的应用程序，移植的过程包括因为两种平台使用的技术不同而进行的改变。移植的难易程度依赖于应用程序的类型、复杂程度和要实现的功能。大量使用 Windows 特定技术（比如 ADO）的应用程序比单纯使用 Delphi 数据库技术的应用程序更难于移植。

将 Windows/VCL 数据库应用程序移植到 Kylix/CLX 应用程序的一般步骤如下：

(1) 考虑数据库数据保存的位置。dbExpress 只提供了 Oracle、Interbase、DB2 和 MySQL 等数据库的驱动程序，移植的数据需要保存在这些 SQL 服务器上。

通过 Delphi 提供的 Data Pump(见第 5 章)工具，可以将本地数据库数据(比如 Paradox、dBase 和 FoxPro 等)转移到 dbExpress 支持的数据库中。

(2) 如果已经开发的应用程序中没有将用户界面和包含数据集及数据库连接组件的数据模块分开，则在开始移植前需要将它们分离，这样就可以将移植时需要使用的一系列新的组件与以前组件的数据模块隔离开。然后将代表用户界面的表单经过简单修改，就可以直接移植到新的应用程序中。

**注意：**以下各个步骤假设数据集和连接组件都放置在单独的数据模块中。

(3) 在应用程序中建立一个新数据模块来放置 CLX 版本的数据集和连接组件。

(4) 为原始应用程序中的每个数据集，添加一个 dbExpress 数据集、TDataSetProvider 和 TclientDataSet 组件，根据组件等级的区别决定使用哪一个 dbExpress 数据集，并给这些组件一个有意义的名字。

- 将 TclientDataSet 组件的 ProviderName 属性设置为 TDataSetProvider 组件的名字。
- 将 TDataSetProvider 组件的 DataSet 属性设置为 dbExpress 数据集。
- 修改任何引用原始数据集的数据源的 DataSet 属性，以便它现在可以引用客户数据集。

(5) 设置新数据集的属性来匹配原始的数据集。

- 如果原始的数据集是 TTable、TADOTable 或 TIBTable 组件，则设置新的 TSQLTable

数据集的 `TableName` 属性为原始属性集中 `TableName` 属性的内容，同时也要拷贝那些用于设置主从表关系或指定索引的所有属性。限定数据范围和过滤（filter）的属性应该设置为该客户数据集，而不是新的 `TSQLTable` 组件。

- 如果原始的数据集 `TQuery`、`TADOQuery` 或 `TIBQuery` 等组件，则将新的 `TSQLQuery` 组件的 `SQL` 属性设置为原始数据集的 `SQL` 属性，设置 `TSQLQuery` 的 `Params` 属性匹配原始数据集中 `Params` 或 `Parameters` 属性的值。如果已经设置了 `DataSource` 属性以建立主从关联关系，则也应该拷贝该属性。
- 如果原始的数据集 `TStoredProc`、`TADOStoredProc` 或 `TIBStoredProc` 等组件，则将新的 `TSQLStoredProc` 组件的 `StoredProcName` 属性设置为原始组件的 `StoredProcName` 或 `ProcedureName` 属性，设置 `TSQLStoredProc` 组件的 `Params` 属性匹配原始数据集中 `Params` 或 `Parameters` 属性的值。

（6）与原始应用程序的任何数据库连接组件（`Tdatabase`、`TIBDatabase` 或 `TADOConnection`）对应，应在新建立的数据模块中添加一个 `TSQLConnection` 组件，并且必须为每一个连接的数据库服务器添加一个 `TSQLConnection` 组件（如果它们没有连接组件）。例如，可以在 ADO 数据集中使用 `ConnectionString` 属性，或将一个 BDE 数据集的 `DatabaseName` 属性设置为 BDE 别名。

（7）对于第四步添加的数据集，设置它的 `SQLConnection` 属性为对应数据库连接的 `TSQLConnection` 组件。

（8）在每个 `TSQLConnection` 组件中，指定需要建立数据库连接的信息。可以在 `TSQLConnection` 组件上面双击鼠标，则会显示出 `Connection Editor`（连接编辑器），在其中编辑参数值以便指明合适的设置。如果已经在第一步中将数据传送到了一个新的数据库服务器中，则应指定适合于新服务器的设置。如果仍使用以前的服务器，则可以在原始的连接组件中查找这类信息。

- 如果原始应用程序使用 `TDatabase`，则必须转移显示在 `Params` 和 `TransIsolation` 属性中的信息。
- 如果原始应用程序使用 `TADOConnection`，则必须转移显示在 `ConnectionString` 和 `IsolationLevel` 属性中的信息。
- 如果原始应用程序使用 `TIBDatabase`，则必须转移显示在 `DatabaseName` 和 `Params` 属性中的信息。
- 如果没有原始连接组件，则必须转移与 BDE 别名或显示在数据集的 `ConnectionString` 属性中相关的信息。

可以在一个新的连接名字下保存这些参数的设置。

5. 修改 dbExpress 应用程序的数据 dbExpress 应用程序使用客户数据集支持编辑功能。当将编辑的结果提交到一个客户数据集时，修改的数据会写入到该客户数据集放在内存中的数据快照中，但是不能自动写入到数据库服务器中。如果你原始的应用程序使用一个客户数据集来缓存修改的数据，则不需要进行任何改变就可以支持在 Linux 下的编辑。但是，如果原始应用程序中依赖于 Windows 上大多数数据集的缺省行为，它们在添加记录时会修改的数据写入到数据库服务器中，则现在必须进行一些修改以便满足客户数据集的要求。

可以使用两种方法转换以前没有使用缓冲区保存修改数据的应用程序：

(1) 可以通过编写代码模仿 Windows 中数据集的行为，即只要提交数据，就将更新的记录写入到数据库服务器中。要实现这一点，可以为客户端数据集设置一个 AfterPost 事件，在该事件中编写代码将每次的数据更新都写入到数据库服务器中。代码如下：

```
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
    with DataSet as TClientDataSet do
        ApplyUpdates(1);
    end;
```

(2) 调整用户界面来处理缓冲区中的数据更新。这种方法有特定的好处，比如可以减少网络的流量，并减少处理事务所用的时间。但是，如果切换到使用缓存更新时，必须决定何时将这些更新的数据写入到数据库服务器中，并且在用户界面上要进行一些修改以便让用户能够初始化应用程序的更新，或者向他们提供修改的数据已经写入到数据库服务器中的反馈信息。另外，由于在用户提交一条记录时无法检测到修改的错误，你也需要修改它以便将错误报告给用户，这样他们可以看到哪些更新导致了错误，以及出现了什么类型的错误。

如果原始的应用程序使用 BDE 或 ADO 提供的缓冲区修改，则需要对代码进行适当的修改，以便可以使用一个客户端数据集。BDE 和 ADO 数据集支持缓冲区修改的属性、事件和方法见表 13-9，该表中还列出了 TClientDataSet 中对应的属性、方法和事件。

表 13-9 BDE 和 ADO 数据集及 TClientDataSet 对应的支持缓冲区修改的属性、事件和方法

BDE 数据集（或 TDatabase）	ADO 数据集	TClientDataSet	作用
CachedUpdates	LockType	不需要，客户端数据集总是提供缓冲区更新的功能	决定缓冲更新是否起作用
不支持	CursorType	不支持	指定如何隔离数据集来自服务器的改变
UpdatesPending	不支持	ChangeCount	指示本地的缓冲是否包含需要写入到数据库中的修改记录
UpdateRecordTypes	FilterGroup	StatusFilter	当写入缓冲区的修改时，指示可见的修改记录的类型
UpdateStatus	RecordStatus	UpdateStatus	指示一条记录是否进行了修改、更新、插入或删除
OnUpdateError	不支持	OnReconcileError	在逐条记录的基础上处理修改错误的实践
ApplyUpdates （在数据集或数据库中）	UpdateBatch	ApplyUpdates	将本地缓冲区中的记录写入到数据库中
CancelUpdates	CancelUpdates 或 CancelBatch	CancelUpdates	删除本地缓冲区中未用的修改，不会将它们写入到数据库中
CommitUpdates	自动处理	Reconcile	在应用程序成功应用修改数据后删除缓冲区中的内容

(续表)

BDE 数据集 (或 TDatabase)	ADO 数据集	TClientDataSet	作用
FetchAll	Not supported	GetNextPacket (和 PacketRecords)	将数据库记录拷贝到本地的缓冲区中用于编辑和修改
RevertRecord	CancelBatch	RevertRecord	如果修改的数据还没有写入到数据库中, 则取消对当前记录的修改

## 13.6 建立跨平台 Internet 应用程序

1. 建立 Internet 应用程序 Internet 应用程序使用标准 Internet 协议连接客户和服务器的 Client/Server 结构, 用于 client/server 通信, 从而可以使应用程序实现跨平台。例如, 一个 Internet 应用程序的服务器端程序通过该计算机的 Web 服务器软件与该客户通信, 该服务器应用程序一般是单独运行在 Linux 或者 Windows 上。

可以使用 Delphi 或 Kylix 建立 Web 服务器应用程序, 比如在 Linux 上发布的 CGI 或 Apache 应用程序。在 Windows 上, 也可以建立其他类型的 Web 服务器, 比如微软的服务器动态库 (ISAPI)、Netscape 的服务器动态库 (NSAPI) 和 Windows CGI 应用程序等。只有使用 Web Broker 的 CGI 应用程序或其他应用程序可以同时运行在 Windows 或 Linux 平台上。

2. 将 Internet 应用程序移植到 Linux 上 如果希望将已经建立的 Internet 应用程序修改为跨平台的, 则应考虑是否需要移植 Web 服务器, 或者是否希望在 Linux 上建立一个新的应用程序。如果应用程序使用了 Web Broker, 并且写入到 Web Broker 界面而没有使用本地的 API, 则该应用程序移植到 Linux 上就不是很难。

如果应用程序中使用了 ISAPI、NSAPI、Windows CGI 或其他 Web API, 则很难移植到 Linux 上。因为在移植时需要查找所有的源文件并将这些 API 调用转换成 Apache 或 CGI 调用。同时还需要进行其他修改。

## 第 14 章 跨平台应用程序实例

由于以前使用 Delphi 开发的应用程序只能运行在 Windows 下,所以如果希望这些应用程序也能运行在 Linux 平台上,则需要通过 CLX 组件对其进行移植。通过移植可以保留以前的工作成果,减少开发费用及开发周期,同时又能保证具有相同的用户界面及功能,为用户提供方便。所以,掌握 Delphi 6 的应用程序移植方法至关重要。

### 14.1 BDE 应用程序移植

使用 BDE 组件建立运行在 Windows 平台上的数据库应用程序,并且通过使用 dbExpress 组件,可将其移植为同时运行在 Windows 和 Linux 平台上的跨平台数据库应用程序。通过对该应用程序的移植,可以看出 VCL 和 CLX 开发数据库应用程序的异同点。

在该实例中,还讲述了使用 TDBGrid 组件的一些常用技巧,比如字段的下拉列表输入、通过其他基表字段的输入、弹出一个窗口进行输入等。另外,也通过该实例说明了如何设置基表中字段对象的一些属性,并通过字段编辑器对其进行管理。

#### 14.1.1 使用 BDE 开发

该应用程序的实现步骤如下:

1. 新建一个应用程序 使用 File|New|Application 新建一个应用程序,此时会自动建立一个主表单 Form1,其对应的单元文件是 Unit1。

2. 添加数据模块和表单 使用 New|Data Module 菜单向该应用程序中添加一个数据模块 DataModule2,其对应的单元文件是 Unit2;再选择 New|Form 菜单,向该应用程序中添加第二个表单 Form3,其对应的单元文件是 Unit3。

3. 单元文件的相互引用 由于 Form1 需要使用 DataModule2 和 Form3,所以选择 Form1 后,再选择 File|Use Unit 菜单,在打开的单元文件管理面板中选择 Unit2 和 Unit3,然后将其添加到 Unit1 中;同样,Form3 也需要使用 uses 语句引用 Form1 和 DataModule2 对应的单元文件 Unit1 和 Unit2。

4. 在数据模块中添加组件并设置组件属性 在数据模块窗口添加两个数据源 DataSource1 和 DataSource2;再通过 BDE 面板的 TTable 组件添加两个基表 Table1 和 Table2;最后通过 BDE 面板向数据模块中添加一个数据库组件 Database1。接下来设置各个组件的属性,数据模块中各个组件属性的设置值见表 14-1。数据模块窗口的显示如图 14-1 所示。

表 14-1 设置数据模块中各个组件的属性

组件	属性	设置值	说明
Database1	AliasName	INTRBASE1	使用的数据库别名
	Connected	True	



(续表)

组件	属性	设置值	说明
	DatabaseName	db1	设置其他组件使用的数据库名称
	LoginPrompt	False	不弹出默认注册窗口
	Params	USER NAME:sysdba PASSWORD:masterkey	设置默认的用户名和口令
Table1	Active	True	
	DatabaseName	db1	
	TableName	PROJ_DEPT_BUDGET	
Table2	Active	True	
	DatabaseName	db1	
	TableName	EMPLOYEE_PROJECT	
DataSource1	DataSet	Table1	
DataSource2	DataSet	Table2	

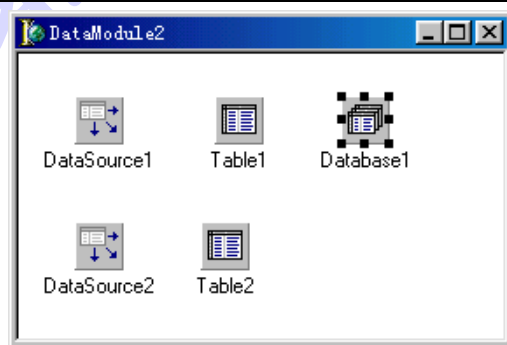


图 14-1 数据模块窗口的显示

5. 在主表单中添加组件 通过 Data Controls 面板，分别在 Form1 中添加一个数据表格组件 DBGrid1 和一个数据导航组件 DBNavigator1。将这两个组件的 DataSource 属性都设置为 DataModule2.DataSource1，此时，Form1 的显示如图 14-2 所示。

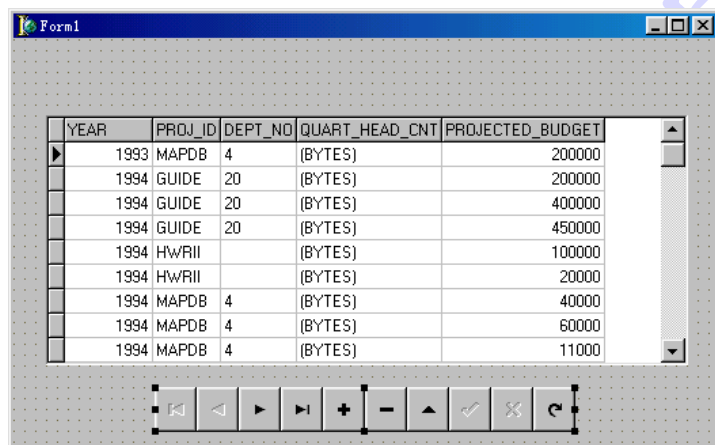


图 14-2 Form1 添加组件后的显示



6. 在 Form3 中添加组件 在表单 Form3 中添加一个多文本组件 Memo1、两个带图表的按钮组件 BitBtn1 和 BitBtn2。将 Memo1 的 Lines 属性中的设置值删除；将 BitBtn1 和 BitBtn2 的 Kind 属性分别设置为 bkOK 和 bkCancel，将 Captions 属性分别设置为“确定”和“取消”，并设置合适的字体。此时，Form3 的显示如图 14-3 所示。该表单用于显示和输入 Table1 中 PROJECTED\_BUDGET 字段的数据。

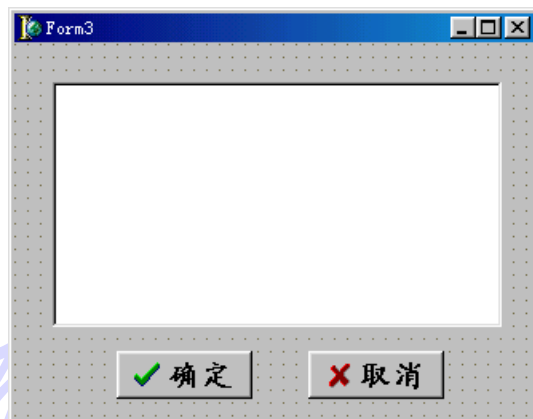


图 14-3 Form3 的显示

7. 在字段编辑器中添加字段 打开 DataModule2，然后在 Table1 上面单击鼠标右键，从弹出的上下文菜单中选择 Fields Editor，则会打开 Table1 对应的字段编辑器，在该字段编辑器中单击鼠标右键，从弹出的上下文菜单中选择 Add All Fields，则 Table1 中所有的字段都添加到了该编辑器中，此时该编辑器的显示如图 14-4 所示。此时在 DataModule2 属性面板的对象列表中也会列出编辑器中这些字段对应的对象，对象名称是表名与字段名的结合体。比如，DEPT\_NO 字段对应的对象是 Table1DEPT\_NO，可以通过该对象设置字段的显示属性。

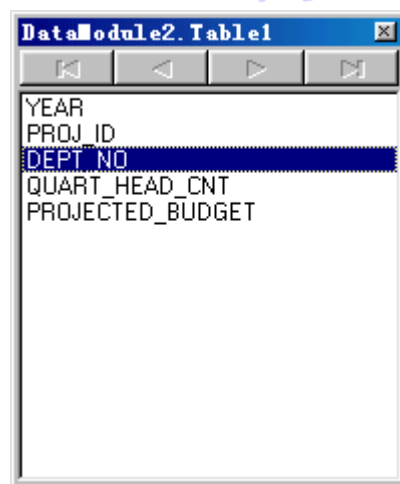


图 14-4 Table1 对应的字段编辑器添加了所有字段

8. 使 DEPT\_NO 字段通过另一个基表的字段输入数据 在字段编辑器中选择 DEPT\_NO，或者打开 DataModule2 对应的属性面板，单击面板顶部的下拉框，从

DataModule2 的对象列表中选择 Table1DEPT\_NO，即 DEPT\_NO 字段对应的对象，然后为该对象设置属性，见表 14-2。

表 14-2 设置字段 DEPT\_NO 的属性

对象名称	属性	设置值	说明
Table1DEPT_NO	DisplayLabel	DEPT_NO	字段显示标签
	FieldKind	fkLookup	设置字段类型为查询字段，即表示通过其他数据源的字段输入数据
	FieldName	DEPT_NO	对应的字段名称
	KeyFields	PROJ_ID	当前基表的关键字段
	LookupDataSet	Table2	提供原始数据的数据集
	LookupKeyFields	PROJ_ID	原始数据集与当前数据集对应的关键字段
	LookupResult	EMP_NO	该查询结果字段作为当前字段的数据来源
	Name	Table1DEPT_NO	字段对象的名称

运行应用程序后，单击 DEPT\_NO 字段后，会弹出一个下拉框，显示出另一个数据集 Table2 中对应字段提供的数据，如图 14-5 所示。该数据是根据 Table2 中 PROJ\_ID 字段的值作为关键字查询出来的。当从列表中选择了一个值后，Table1 中的 PROJ\_ID 和 DEPT\_NO 两个字段的值会同时改变，因为 Table1 中的 PROJ\_ID 与 Table2 中的 PROJ\_ID 字段的值相对应。

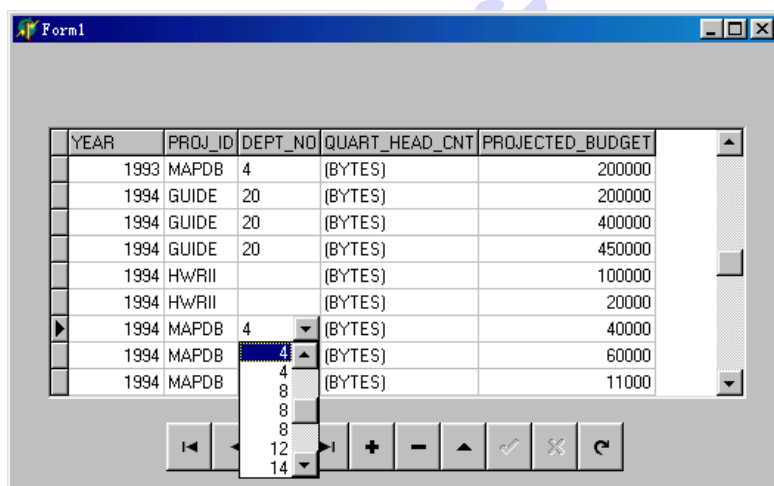


图 14-5 通过另一个数据集中的字段来输入一个字段的值

9. 在 DBGrid1 的编辑器中添加显示的字段 默认情况下，DBGrid1 会显示 Table1 中所有的字段，但是无法控制各个列（字段）的显示属性。如果要控制 DBGrid1 中各个列的显示属性，则需要将这些字段添加到 DBGrid1 的列编辑器中。

在 DBGrid1 的列编辑器中添加字段的方法如下：在 DBGrid1 上面单击鼠标右键，从弹出的上下文菜单中选择 Columns Editor 菜单，此时会打开列编辑器。在打开的列编辑器中

再单击鼠标右键，从弹出的上下文菜单中选择 Columns Editor 菜单 Add All Fields，表示将基表的所有字段都添加到 DBGrid1 中，这样在列编辑器中会显示出所有列的名字，其显示如图 14-6 所示。默认情况下，所有列的名字与其对应的字段的名称都相同。

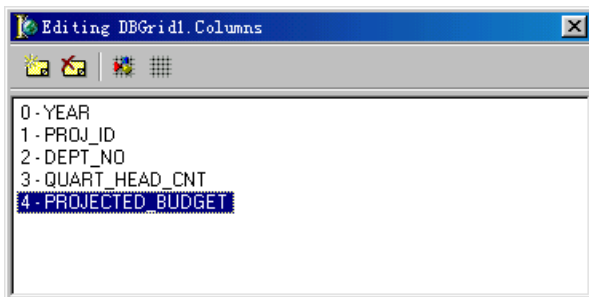


图 14-6 DBGrid1 对应的列编辑器

当在列编辑器中选择一个列，并打开对象观察器的属性设置面板时，可以修改该列的属性。一些重要的属性如下：

- **Alignment** 设置一列中数据的对齐方式，**taLeftJustify** 表示靠左，**taRightJustify** 表示靠右，**taCenter** 表示居中。
- **ButtonStyle** 设置该列是否显示一个下拉框或按钮。当设置 **cbsAuto** 值时，如果当前列的 **PickList** 属性设置了一些数据或者该列对应字段从其他数据集中查询数据进行输入时，会显示一个下拉框；如果设置 **cbsEllipsis** 值，则该列的右边会出现一个按钮，单击该按钮会触发 DBGrid1 的 **OnEditButtonClick** 事件，通过该事件可以打开一个窗口设置该字段的输入数据；**cbsNone** 表示该列不使用下拉框或按钮。
- **DropDownRows** 如果当前列是一个下拉框时，则该字段设置的值用于限制下拉框显示的行数。
- **FieldName** 设置该列对应的字段名称。
- **PickList** 如果一个列有一些固定值，则可以通过为该属性设置一些固定值，让用户从列表中选择值进行输入。单击该属性右边的按钮，可以打开一个字符串列表编辑器，可以在该编辑器中设置一些值，每个值占一行，这样在输入数据时，可以通过一个下拉框选择这些值。
- **Title** 设置列标题的一些属性，包括以下子属性：
  - ◇ **Alignment** 设置列标题的对齐方式，**taLeftJustify** 表示靠左；**taRightJustify** 表示靠右；**taCenter** 表示居中。
  - ◇ **Caption** 设置标题显示的内容，可以设置为中文标题。
  - ◇ **Color** 设置标题的背景颜色。
  - ◇ **Font** 设置标题使用的字体。
- **Visible** 设置该列是否可见。
- **Width** 设置该列的宽度。

下面通过属性面板，分别将 YEAR 和 PROJECTED\_BUDGET 列设置一个下拉框和一个按钮。

(1) 使 YEAR 列通过选择固定值进行输入。打开列编辑器，从中选择列 YEAR，然

后打开属性设置面板，单击 **PickList** 属性右边的按钮打开字符串编辑器，在编辑器中输入 1993-2001 年度的值，如图 14-7 所示。单击 OK 按钮确认后关闭该窗口。

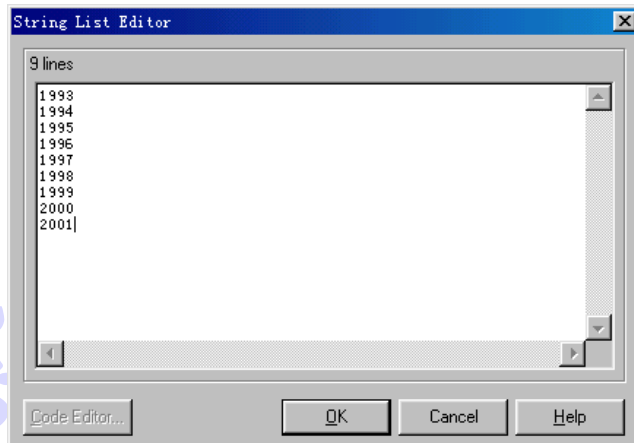


图 14-7 设置 YEAR 列的 PickList 属性

当运行应用程序时，该列的右边会出现一个下拉箭头，单击该箭头，则可以打开一个下拉框，列出了 **PickList** 属性中设置的所有值，如图 14-8 所示，可以从该下拉框中选择一个值作为当前字段的输入值。

该属性适合于有固定输入值的字段。

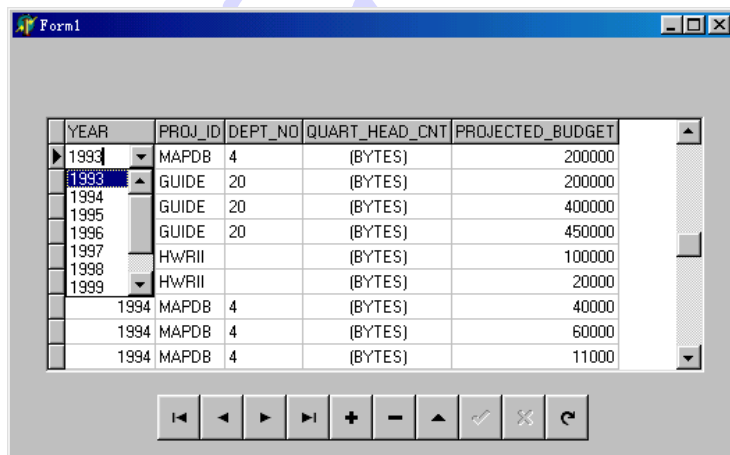


图 14-8 从下拉框中选择输入值

(2) 为 **PROJECTED\_BUDGET** 列设置一个按钮。设置按钮的方法很简单，只要在列编辑器中选择 **PROJECTED\_BUDGET** 列，然后将其 **ButtonStyle** 属性设置 **cbsEllipsis** 即可。此时如果运行程序，则该按钮的显示如图 14-9 所示。

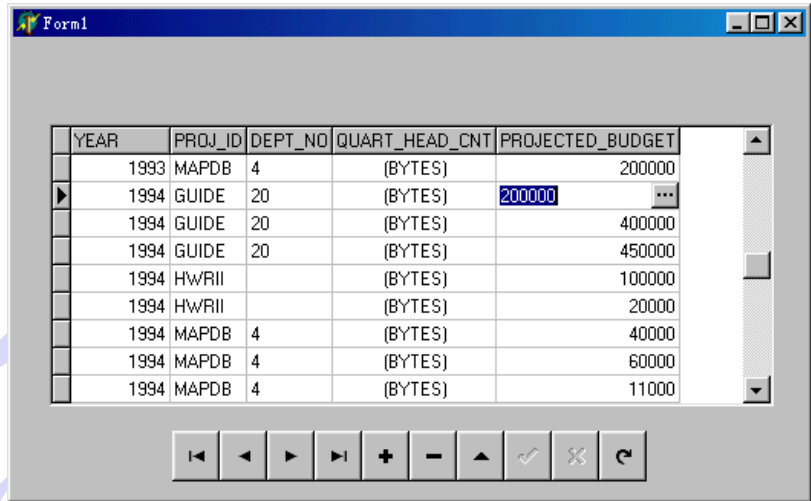


图 14-9 为 PROJECTED\_BUDGET 列设置一个按钮

如果要通过该按钮打开一个窗口输入数据，则需要在 DBGrid1 的 OnEditButtonClick 事件中设置相应的代码。

打开 DBGrid1 对应的对象观察器的 Events 面板，然后在 OnEditButtonClick 一栏双击鼠标，则会打开代码编辑器，然后在代码编辑器中输入相应的代码，完整的程序如下：

```
procedure TForm1.DBGrid1EditButtonClick(Sender: TObject);
begin
    //下面的if语句检验输入焦点是否在Table1PROJECTED_BUDGET字段
    if DBGrid1.SelectedField=DataModule2.Table1PROJECTED_BUDGET then
    begin
        //下面的语句将PROJECTED_BUDGET字段的内容显示在Memo1中
        Form3.Memo1.Text:=DataModule2.Table1PROJECTED_BUDGET.AsString;
        //显示为PROJECTED_BUDGET字段输入数据的表单Form3
        Form3.ShowModal;
    end;
end;
```

在 Form1 对应的单元文件 Unit1 中输完上面的代码后，现在运行程序并单击 PROJECTED\_BUDGET 一系列的按钮，则会打开 Form3。

10. 为 Form3 的按钮添加触发事件。由于 Form3 用于输入 PROJECTED\_BUDGET 字段的数据，所以我们希望单击“确定”按钮后，可以将用户在该窗口的 Memo1 中输入的数据保存到 PROJECTED\_BUDGET 字段中；如果单击“取消”按钮，则放弃输入的数据并关闭该输入数据窗口。

由于 PROJECTED\_BUDGET 字段只能输入数值，但是用户在表单 Form3 中可能会误输入字符，所以此时需要进行异常处理，以保证用户输入正确的数据。

(1) 添加“确定”按钮的代码。在 Form3 的“确定”按钮 BtnBit1 上面双击鼠标，则会打开代码编辑器，然后输入相应代码，完整的程序代码如下：

```
procedure TForm3.BitBtn1Click(Sender: TObject);
var
    i:integer;
begin
```

```

try
    i:=StrToInt(Memo1.Text);    //如果输入了字符，则会触发一个异常
    DataModule2.Table1.Edit;    //将基表设置为可编辑状态，否则无法插入数据
    //下面的语句将用户输入的数据插入到字段中
    DataModule2.Table1PROJECTED_BUDGET.AsString:= Memo1.Text;
    close;    //关闭表单Form3
except
    //Memo1中输入的不是数字时弹出该对话框
    on EConvertError do showMessage('请输入数字值! ');
end;
end;

```

(2) 添加“取消”按钮的代码。在 Form3 的“取消”按钮 BtnBit2 上面双击鼠标，则会打开代码编辑器，然后输入相应代码，完整的程序代码如下：

```

procedure TForm3.BitBtn2Click(Sender: TObject);
begin
    close;    //关闭表单Form3
end;

```

11. 完整的源代码 DataModule2 对应的 Unit2 单元文件中的代码全部是默认值，没有进行任何修改，所以在此不再列出。下面只列出 Form1 对应的 Unit1 单元文件和 Form3 对应的 Unit3 单元文件的源代码。

(1) Unit1 单元文件源代码如下：

```

unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, ExtCtrls, DBCtrls, Grids, DBGrids;

type
    TForm1 = class(TForm)
        DBGrid1: TDBGrid;
        DBNavigator1: TDBNavigator;
        procedure DBGrid1EditButtonClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

uses Unit2, Unit3;    //对其他两个单元文件的引用

{$R *.dfm}

```



```

procedure TForm1.DBGrid1EditButtonClick(Sender: TObject);
begin
    .....//略，见本章前面讲述的该事件的代码
end;

end.

```

(2) Unit3 单元文件源代码如下：

```

unit Unit3;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Buttons;

type
    TForm3 = class(TForm)
        Memo1: TMemo;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
var
    Form3: TForm3;

implementation

uses Unit1, Unit2;          //引用其他两个单元文件

{$R *.dfm}

procedure TForm3.BitBtn1Click(Sender: TObject);
var
    i: integer;
begin
    .....// 略，见本章前面讲述的该事件的代码
end;

procedure TForm3.BitBtn2Click(Sender: TObject);
begin
    close;
end;
end.

```

### 14.1.2 使用 dbExpress 组件

由于 dbExpress 面板上的组件建立的数据库应用程序既可以运行在 Windows 环境下，又可以运行在 Linux 环境下，所以可以使用该面板的组件来移植以前使用 BDE、ADO 面板的组件开发的只能运行在 Windows 平台下的应用程序。

下面将前面使用 BDE 组件建立的数据库应用程序移植到使用 dbExpress 组件的数据库应用程序。在进行数据库应用程序移植时，一般只需要修改数据模块中不可见的数据库组件，以及与这些组件有关的代码，而不需要修改应用程序的界面及对应的代码。该应用程序的移植步骤如下：

1. 打开文件 首先备份以前建立的 BDE 应用程序，然后重新打开它，再通过 View/Forms 菜单打开 DataModule2 窗口。

2. 删除组件 在 DataModule2 窗口删除 BDE 数据库组件 Table1、Table2 和 Database1。

3. 在 DataModule2 中添加组件 打开 dbExpress 组件面板，然后在数据模块中添加如下组件：用数据库连接组件 SQLConnection1 代替 Database1；用数据集组件 SQLClientDataSet1 代替 Table1；再添加一个数据集组件 SQLTable1 来代替 Table2。

但是由于 SQLTable1 是一个单方向的数据集，不能直接作为字段查询输入的数据集，即不能在 LookupDataSet 中设置它，所以打开 Data Access 组件面板，分别在数据模块中添加一个客户数据集组件 ClientDataSet1 和一个数据集提供者组件 DataSetProvider1，以便改变 SQLTable1 提供的数据的单方向性，使数据可编辑。

**注意：**实际上使用一个 TSQLClientDataSet 组件就可以代替 SQLTable1、ClientDataSet1 和 DataSetProvider1 三个组件，但是本实例使用这些组件的目的在于提供尽可能多的方法。

以上所有组件添加完毕后，DataModule2 的显示如图 14-10 所示。

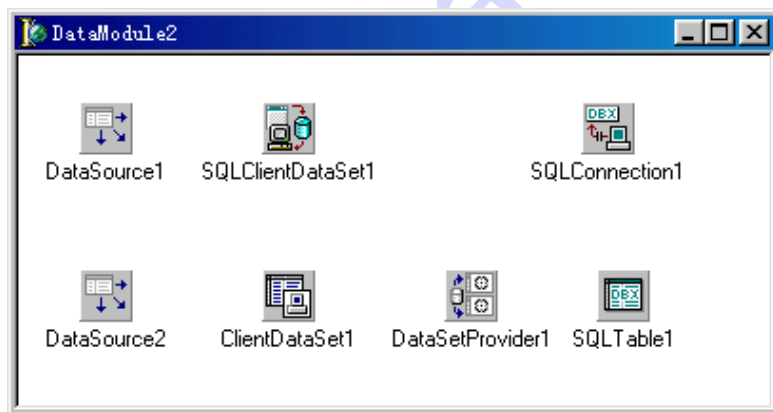


图 14-10 添加数据库组件后 DataModule2 的显示

4. 设置 DataModule2 中各个组件的属性 它们的属性设置值见表 14-3。

表 14-3 设置 DataModule2 窗口组件的属性

组件	属性	设置值	说明
SQLConnection1	Connected	True	
	ConnectionName	Interbase	设置一个已经建立的数据库连接的名字
	KeepConnection	True	
	LibraryName	d:\borland\delphi6\bin\dbexpint.dll	设置 InterBase 数据库对应的动态库 dbexpint.dll 的实际路径，该路径与 Delphi 安装的目录有关
	LoginPrompt	False	不显示默认口令注册窗口
	Params	(见图 14-11)	设置连接参数，包括用户名及口令
SQLClientDataSet1	Active	True	
	CommandType	ctTable	表示该数据集是一个基表
	CommandText	PROJ_DEPT_BUDGET	基表的名称
	DBConnection	SQLConnection1	数据库连接的名字
SQLTable1	SQLConnection	SQLConnection1	
	TableName	EMPLOYEE_PROJECT	
DataSetProvider1	DataSet	SQLTable1	
ClientDataSet1	Active	True	
	ProviderName	DataSetProvider1	
DataSource2	DataSet	ClientDataSet1	
DataSource1	DataSet	SQLClientDataSet1	

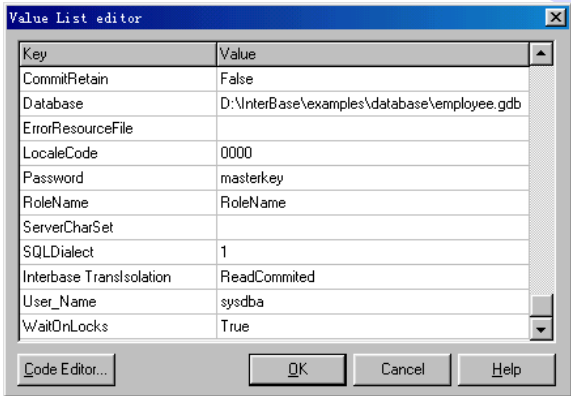


图 14-11 SQLConnection1 组件的 Params 属性对应的设置面板

5. 设置 DEPT\_NO 字段对象的属性 由于 DEPT\_NO 字段需要通过其他数据集进行输入, 所以需要重新设置其属性。设置方法与 BDE 中的相同。具体步骤如下: 在 SQLClientDataSet1 组件上单击鼠标并选择 Fields Editor 菜单, 打开字段编辑器, 并添加所有的字段。然后在该字段编辑器中选择 DEPT\_NO, 或者打开 DataModule2 对应的属性面板, 单击面板顶部的下拉框, 从 DataModule2 列表中选择 SQLClientDataSet1DEPT\_NO, 即 DEPT\_NO 字段对应的对象, 该对象大部分属性可以使用表 14-2 中的设置值, 但是下列个别属性需要改变: LookupDataSet 属性改为 ClientDataSet1, Name 属性也相应地变为 SQLClientDataSet1DEPT\_NO。

6. 修改代码 由于该应用程序使用 SQLClientDataSet1 数据集代替了 Table1 数据集, 所以代码中与 Table1 有关的内容, 都应修改为 SQLClientDataSet1。而其他数据库组件没有相关的代码, 所以不需要修改。

修改后的代码如下:

(1) Unit1 中修改后的部分源代码:

```
procedure TForm1.DBGrid1EditButtonClick(Sender: TObject);
begin
  if DBGrid1.SelectedField=DataModule2.SQLClientDataSet1PROJECTED_BUDGET
  then
  begin
    //下面的语句中将Table1修改为SQLClientDataSet1
    Form3.Memo1.Text:=DataModule2.SQLClientDataSet1PROJECTED_BUDGET.AsString
    ;
    Form3.ShowModal;
  end;
end;
```

(2) Unit2 中修改后的部分源代码:

```
procedure TForm3.BitBtn1Click(Sender: TObject);
var
  i:integer;
begin
  try
    i:=StrToInt(Memo1.Text);
    //下面的两个语句将Table1修改为SQLClientDataSet1
    DataModule2.SQLClientDataSet1.Edit;
    DataModule2.SQLClientDataSet1PROJECTED_BUDGET.AsString:= Memo1.Text;
    close;
  except
    on EConvertError do showMessage('请输入数字值! ');
  end;
end;
```

至此, 整个应用程序移植完毕。现在运行该应用程序, 与使用 BDE 组件时的执行结果会完全相同。可见, 数据库应用程序移植时, 主要修改了一些数据库组件及与数据库有关的代码, 而数据库的界面基本不需要修改。

该应用程序如果要在 Linux 平台上运行, 只需要使用 Kylix 重新编译即可。

## 14.2 计 时 器

Delphi 在 CLX 应用程序的 Additional 面板，提供了一个计时器组件 TLCDNumber，通过该组件可以精确地控制时间，它可以将时间精确度控制在 1 毫秒。下面的实例将通过该组件设计一个计时器，该实例可以在 Delphi 6 的 Demo 目录的 CLX 子目录下找到。

该实例的实现步骤如下：

1. 使用 File|New|CLX Application 菜单新建一个跨平台的应用程序。
2. 将表单 Form1 适当缩小，然后在表单中添加下列组件：通过 Additional 面板的 TLCDNumber 组件添加一个计时组件为 LCDNumber1，再通过 TTimer 组件添加一个定时器 Timer1。然后通过 Startard 面板的 TBotton 组件添加三个按钮 Botton1、Botton2、Botton3。
3. 设置各个组件的属性。其设置值见表 14-4。

表 14-4 设置各个组件的属性

组件	属性	设置值	说明
LCDNumber1	Align	alTop	靠表单顶部放置
	Anchors	[akLeft,akTop,akRight]	左边、右边和上边与表单的对应边对齐
	ParentColor	False	与表单颜色不同
	SegmentStyle	ssFilled	数字的显示风格为填充形式
	Value	00:00.000	设置使用的格式
Timer1	Interval	1	设置触发定时器的时间间隔为 1 毫秒
Button1	Caption	开始	
Button2	Caption	停止	
Button3	Caption	复位	

以上各个组件的属性设置完毕后，其界面如图 14-12 所示。



图 14-12 计时器的设计界面

4. 设置触发事件并添加代码。为三个按钮分别设置一个 OnClick 事件；为定时器设置一个 OnTimer 事件，表示到达 Interval 设置的时间间隔就触发该事件；为 LCDNumber1 设置一个 OnMouseUp 事件，当在 LCDNumber1 上面按下鼠标又释放时触发该事件，用于设置 LCDNumber1 边框的不同显示风格。另外，还需要定义一个过程来设置属性。

综合上述事件及过程，表单 Form1 对应的 Unit1 单元文件的完整源代码如下：

```
unit main;
```

```

interface

uses
  SysUtils, Types, Classes, QGraphics, QControls, QForms, QDialogs,
  QStdCtrls, QTypes, QExtCtrls;

type
  TForm1 = class(TForm)
    LCDNumber1: TLCDNumber;
    Timer1: TTimer;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    LCDNumber2: TLCDNumber;
    procedure Timer1Timer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure LCDNumber1MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
    FRunning: Boolean; //该私有变量用于读取属性值
    procedure SetRunning(Value: Boolean); //该过程用于设置属性值
  public
    { Public declarations }
    StartTime: Extended; //下面三个语句设置公用变量
    ElapsedTime: Extended;
    Reset: Boolean;
    property Running: Boolean read FRunning write SetRunning; //定义属性
  end;

var
  Form1: TForm1;

implementation

{$R *.xrm}

procedure TForm1.SetRunning(Value: Boolean); //实现设置属性Running的过程
begin
  if FRunning <> Value then //如果Running值与原来属性值不同, 执行下面的代码
  begin
    FRunning := Value;
    StartTime := ElapsedTime; //开始时间为已经计算的时间
    if Value then Reset := False; //如果FRunning属性值为True, 表示正在运行
  end;
end;

```



```

procedure TForm1.Timer1Timer(Sender: TObject); //每间隔1毫秒就检查一次“复位”
状态
begin
    if Reset then ElapsedTime := Now else ElapsedTime := Now - StartTime; //
复位则重计时
    if Running then //如果正在运行, 则使用全局函数FormatDateTime格式化计时时间
        LCDNumber1.Value := FormatDateTime('nn:ss.zzz', ElapsedTime);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    Running := True;          //表示开始运行
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Running := not Running;   //表示没有运行
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    if not Running then //如果停止运行则将计时器重置为0
    begin
        LCDNumber1.Value := '00:00.000';
        Reset := True;
    end;
end;

procedure TForm1.LCDNumber1MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
var
    NewBorderStyle: Integer;
    NewSegmentStyle: Integer;
begin
    case Button of
        mbLeft: //释放的是鼠标左键时, 改变LCDNumber1边框的风格
            begin
                NewBorderStyle := Ord(LCDNumber1.BorderStyle) + 1;
                if NewBorderStyle > Ord(High(TBorderStyle)) then
                    NewBorderStyle := Ord(Low(TBorderStyle));
                LCDNumber1.BorderStyle := TBorderStyle(NewBorderStyle);
            end;
        mbRight: //释放的是鼠标右键时, 改变LCDNumber1字体的风格
            begin
                NewSegmentStyle := Ord(LCDNumber1.SegmentStyle) + 1;
                if NewSegmentStyle > Ord(High(TLCDSegmentStyle)) then
                    NewSegmentStyle := Ord(Low(TLCDSegmentStyle));
                LCDNumber1.SegmentStyle := TLCDSegmentStyle(NewSegmentStyle);
            end;
    end;
end;

```

```

end;
end;

end.

```

### 14.3 浏 览 器

TTextBrowser 组件是 CLX 应用程序的 Common Controls 组件面板提供的, 用于实现超文本的浏览。本实例就是通过该组件建立了一个简单的超文本浏览器。该实例与位于 Delphi 6 安装目录的 Demo\CLX\TextBrowser 子目录中的项目文件相似, 但是这个 Demo 程序不能打开任意一个文件, 而本实例却可以。

该实例的实现步骤如下:

1. 使用 File|New|CLX Application 菜单新建一个跨平台应用程序。
2. 将表单 Form1 的 Name 属性修改为 TextBrowserForm, Captions 属性修改为“TTextBrowser 组件实例程序”。
3. 在表单中添加下列组件:

使用 Standard 组件面板的 TGroupBox 组件在表单的右上角添加两个组件 GroupBox1 和 GroupBox2。将 GroupBox1 的 Caption 属性设置为“超链接的颜色”; 将 GroupBox2 的 Caption 属性设置为“文本的颜色”。

在 GroupBox1 中放置两个单选按钮组件 RadioButton1 和 RadioButton2, 将其 Caption 属性分别修改为“蓝色”、“红色”, 将字体颜色也分别设置为蓝色和红色, 并将字体都设置为“楷体|加粗”; 在 GroupBox2 中放置两个单选按钮组件 RadioButton3 和 RadioButton4, 将其 Caption 属性分别修改为“黑色”、“绿色”, 将字体颜色也分别设置为黑色和绿色, 并将字体都设置为“楷体|加粗”。

在 GroupBox1 和 GroupBox2 的下面放置三个按钮 Button1、Button2、Button3, 将其 Caption 属性分别修改为“主页”、“向后”、“向前”, 将它们的 Name 属性分别修改为 HomeBtn、BackBtn、ForwardBtn。

在 GroupBox2 的右边放置一个文本编辑框 Edit1, 将其 Name 属性修改为 LinkSourceEdit, 将其 Text 属性中的内容删除。

在 LinkSourceEdit 编辑框的上面放置一个标签 Label1, 将其 Caption 属性修改为“地址:”。

在 LinkSourceEdit 编辑框的右边放置一个 TSpeedButton 组件 SpeedButton1, 并在其 Glyph 属性设置一个位于 Delphi6 安装目录的子目录...\images\buttons 下的 directry.bmp 图像文件作为图标。

在 LinkSourceEdit 编辑框的下面添加一个 TRadioGroup 组件 RadioGroup1, 将其 Caption 属性设置为“文本格式”; 在其 Items 打开的文本编辑框中加入两项“格式化的”、“普通的”; 将 Orientation 属性设置为“orHorizontal”, 表示添加的两个单选按钮水平排列。

在 RadioGroup1 的右边放置一个复选框 CheckBox1, 将其 Name 属性修改为 UnderlineCheckbox; Caption 属性修改为“下划线”; Checked 属性设置为 True。

在复选框 UnderlineCheckbox 的右边, 再放置一个按钮, 将其 Name 属性修改为

ReloadBtn; 将其 Caption 属性修改为“刷新”。

在表单的底部，添加一个状态条 StatusBar。在状态条的上面，放置一个 TTextBrowser 组件 TextBrowser，设置其 Anchors 属性为[akLeft,akTop,akRight,akBottom]，以便根据窗口来改变其大小。

最后放置一个对话框组件 OpenFileDialog1，用于选择 HTML 文件，并设置其 Filter 属性为 \*.html|\*.htm|\*.\*，以便筛选 HTML 文件。

以上所有组件添加完毕并设置了属性后，主表单 TextBrowserForm 的显示如图 14-13 所示。

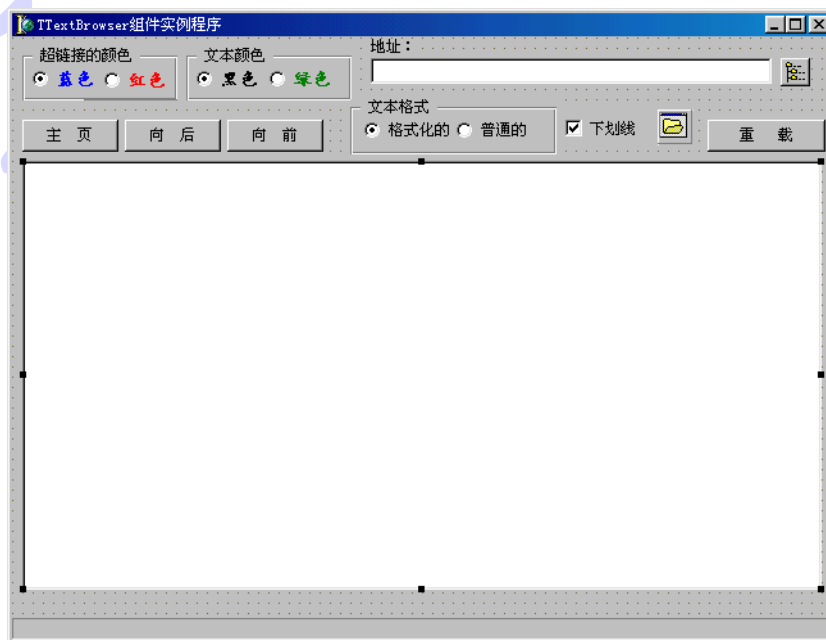


图 14-13 设计完成的主表单 TextBrowserForm

#### 4. 添加代码 该实例主单元文件 main 的源代码如下:

```
unit main;

interface

uses
  SysUtils, Types, Classes, QGraphics, QControls, QForms, QDialogs,
  QComCtrls,
  QStdCtrls, QExtCtrls, QButtons;

type
  TTextBrowserForm = class(TForm)
    HomeBtn: TButton;
    BackBtn: TButton;
    ForwardBtn: TButton;
    TextBrowser: TTextBrowser;
    GroupBox1: TGroupBox;
    RadioButton1: TRadioButton;
```

```

RadioButton2: TRadioButton;
Label1: TLabel;
LinkSourceEdit: TEdit;
ReloadBtn: TButton;
GroupBox2: TGroupBox;
RadioButton3: TRadioButton;
RadioButton4: TRadioButton;
RadioGroup1: TRadioGroup;
UnderlineCheckbox: TCheckBox;
StatusBar: TStatusBar;
SpeedButton1: TSpeedButton;
TextBrowser1: TTextBrowser;
OpenDialog1: TOpenDialog;
    procedure TTextBrowserForm.FormCreate(Sender: TObject);
    procedure HomeBtnClick(Sender: TObject);
    procedure TextBrowserTextChanged(Sender: TObject);
    procedure BackBtnClick(Sender: TObject);
    procedure ForwardBtnClick(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure LinkSourceEditChange(Sender: TObject);
    procedure ReloadBtnClick(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure RadioButton4Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure UnderlineCheckboxClick(Sender: TObject);
    procedure TextBrowserHighlightText(Sender: TObject);
        const HighlightedText: WideString);
    procedure SpeedButton1Click(Sender: TObject);
private
    { Private declarations }
    PathList: TStrings;
public
    { Public declarations }
end;

var
    TextBrowserForm: TTextBrowserForm;

implementation

{$R *.xfm}

procedure TTextBrowserForm.FormCreate(Sender: TObject);
begin
    TextBrowser.UnderlineLink:=UnderlineCheckbox.Checked;
end;

procedure TTextBrowserForm.HomeBtnClick(Sender: TObject); //单击主页按钮
begin

```

```

    TextBrowser.Home;           //返回主页
end;

procedure TTextBrowserForm.TextBrowserTextChanged(Sender: TObject); //打
开不同文件
begin
    //激活按钮“向后”、“向前”
    BackBtn.Enabled := TextBrowser.CanGoBackward;
    ForwardBtn.Enabled := TextBrowser.CanGoForward;
end;

procedure TTextBrowserForm.BackBtnClick(Sender: TObject); //单击“向后”按
钮
begin
    TextBrowser.Backward;
end;

procedure TTextBrowserForm.ForwardBtnClick(Sender: TObject); //单击“向前”
按钮
begin
    TextBrowser.Forward;
end;

procedure TTextBrowserForm.RadioButton1Click(Sender: TObject);
begin
    TextBrowser.LinkColor := clBlue; //选中“蓝色”单选按钮
end;

procedure TTextBrowserForm.RadioButton2Click(Sender: TObject);
begin
    TextBrowser.LinkColor := clRed; //选中“红色”单选按钮
end;

procedure TTextBrowserForm.LinkSourceEditChange(Sender: TObject);
begin
    TextBrowser.FileName := LinkSourceEdit.Text //改变了编辑框的内容(文件名)
end;

procedure TTextBrowserForm.ReloadBtnClick(Sender: TObject);
begin
    TextBrowser.LoadFromFile(TextBrowser.FileName); //重新打开文件
end;

procedure TTextBrowserForm.RadioButton3Click(Sender: TObject);
begin
    TextBrowser.TextColor := clBlack; //选中“黑色”单选按钮
end;

procedure TTextBrowserForm.RadioButton4Click(Sender: TObject);
begin

```

```

    TextBrowser.TextColor := clGreen; //选中“绿色”单选按钮
end;

procedure TTextBrowserForm.RadioGroup1Click(Sender: TObject);
begin
    case RadioGroup1.ItemIndex of
        0: TextBrowser.TextFormat := tfText;           //选中“格式化的”单选按钮
        1: TextBrowser.TextFormat := tfPlainText;      //选中“普通的”单选按钮
    end;
end;

procedure TTextBrowserForm.UnderlineCheckboxClick(Sender: TObject);
begin
    TextBrowser.UnderlineLink := not TextBrowser.UnderlineLink; //“下划线”
    复选框取反
end;

procedure TTextBrowserForm.TextBrowserHighlightText(Sender: TObject;
    const HighlightedText: WideString);
begin
    StatusBar.SimpleText := '链接到: ' + HighlightedText; //显示状态信息
end;

procedure TTextBrowserForm.SpeedButton1Click(Sender: TObject); //单击了
    SpeedButton1
begin
    OpenFileDialog1.Execute;           //打开文件选择框
    LinkSourceEdit.Text := OpenFileDialog1.FileName; //显示文件名
end;

end.

```

5. 运行文件 运行该应用程序后，单击文件选择按钮 **SpeedButton1**，则会弹出如图 14-14 所示的文件选择对话框，从中选择需要的 HTML 文件，再单击“打开”按钮，则会在浏览器中显示该文件，此时如果选择超链接的颜色为“红色”，选择文本的颜色为“绿色”，则主窗口的显示如图 14-15 所示。此时如果选择文本格式为“普通的”，则会显示出打开的 HTML 文件的源文件，如图 14-16 所示。

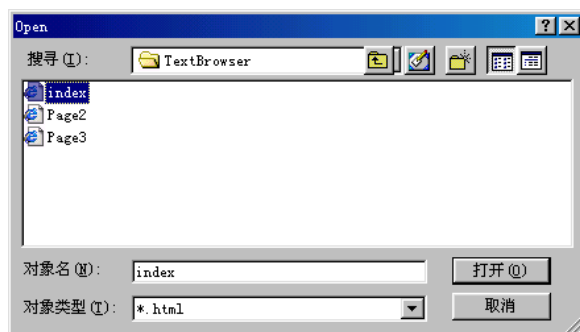


图 14-14 打开文件选择窗口



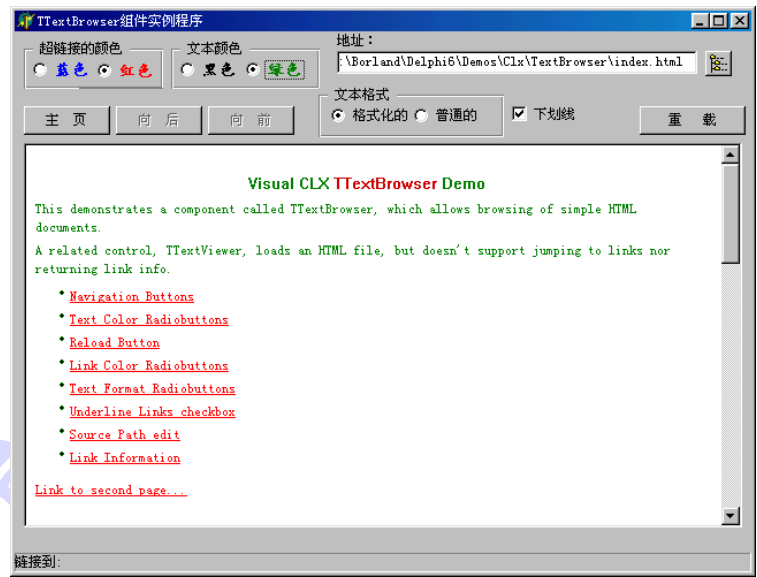


图 14-15 应用程序打开一个 HTML 文件后的显示

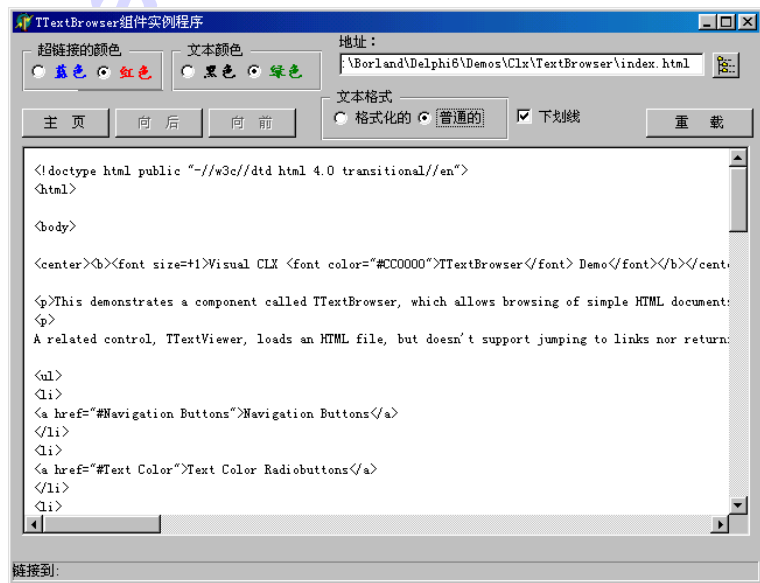


图 14-16 显示 HTML 文件的源文件

## 14.4 多文档应用程序

Delphi 本身提供了一个建立跨平台的多文档应用程序的模板，但是该模板还不是很完善，下面的实例将设计一个完善的多文档应用程序。

该应用程序的实现步骤如下：

1. 先关闭所有的应用程序，然后选择 New|Other 菜单，则会打开对象库面板，选择 Projects 标签页，如图 14-17 所示，在 CLX MDI Application 图标上面双击鼠标，在弹出的目录选择框中选择一个保存新建项目文件的路径，也可以设置一个新的目录，再单击 OK

按钮，则会建立一个新的多文档的应用程序，同时该项目中的所有文件也保存到了设置的目录中。

该应用程序还不是很完善，比如无法保存文件，下面就添加保存文件的功能。

2. 通过 Dialogs 组件面板，在新建立的 MDI 应用程序的主表单上添加一个保存文件对话框 SaveDialog1。此时，应用程序主界面的显示如图 14-18 所示。

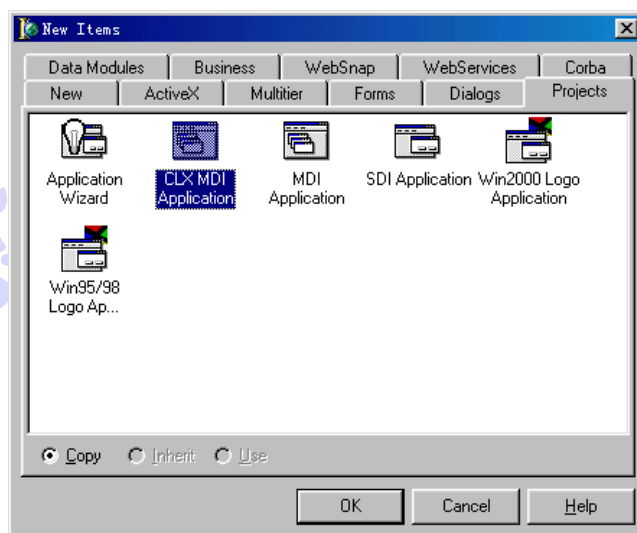


图 14-17 打开对象库的 Projects 标签页

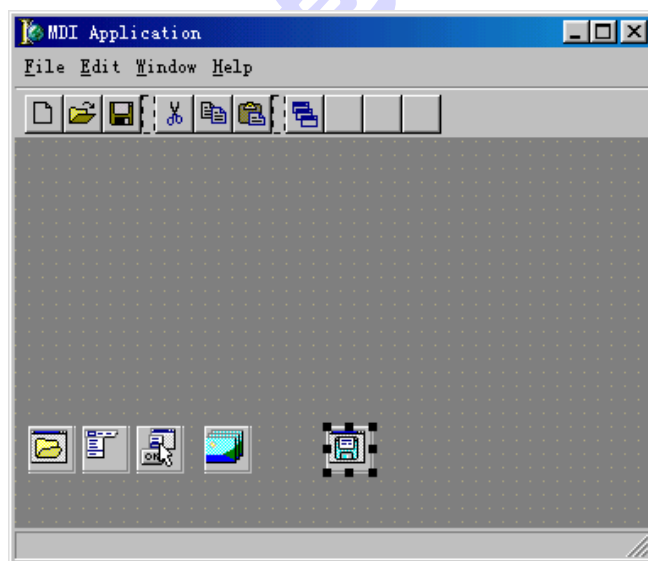


图 14-18 添加了保存文件对话框后的主界面

3. 该应用程序的所有菜单命令都是通过命令列表组件 ActionList1 建立的。所以下面通过它建立保存文件的命令。

在主窗口的命令列表组件的图标上面双击鼠标，则会打开命令列表编辑框，在左边的 Categories 一栏选择 File，则右边的 Actions 一栏相应地会显示出它包括的所有动作，此时 ActionList1 编辑框的显示如图 14-19 所示。

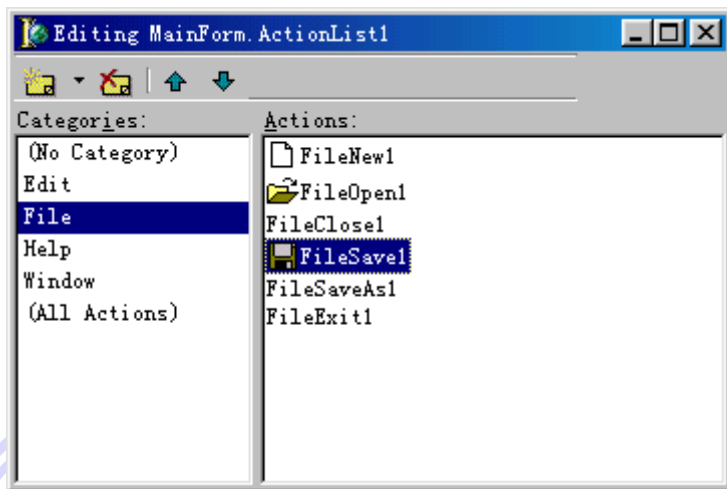


图 14-19 ActionList1 组件的编辑框

在 FileSave1 上面双击鼠标，则会打开代码编辑器，输入的代码如下所示：

```
procedure TMainForm.FileSave1Execute(Sender: TObject);
begin
    if (FileName = 'NONAME' + IntToStr(MDICHildCount)) then //检查文件名是否存在
        FileSaveAs1Execute(Sender) //文件名存在使调用FileSaveAs1命令
    else
        TMDICHild(ActiveMDICHild).Memo1.Lines.SaveToFile(FileName); //保存文件
end;
```

**注意：**FileName 是主表单对应的单元文件中定义的私有变量，类型为 String。另外，在新建立一个文件时，应将使用的文件名保存到 FileName 变量中。

按照同样的方法，为 FileSaveAs1 命令添加如下代码：

```
procedure TMainForm.FileSaveAs1Execute(Sender: TObject);
begin
    SaveDialog1.FileName := ActiveMDICHild.Caption;
    SaveDialog1.InitialDir := ExtractFilePath(FileName);
    if SaveDialog1.Execute then
    begin
        if ActiveMDICHild is TMDICHild then
        begin
            TMDICHild(ActiveMDICHild).Memo1.Lines.SaveToFile(SaveDialog1.FileName);
            FileName := SaveDialog1.FileName;
            ActiveMDICHild.Caption := SaveDialog1.FileName;
            StatusBar.Panels[0].Text := FileName;
        end;
    end;
end;
```

4. 修改快捷按钮的图标。由于所有图标都是通过 ImageList1 组件管理的，该组件编辑器的显示如图 14-20 所示。

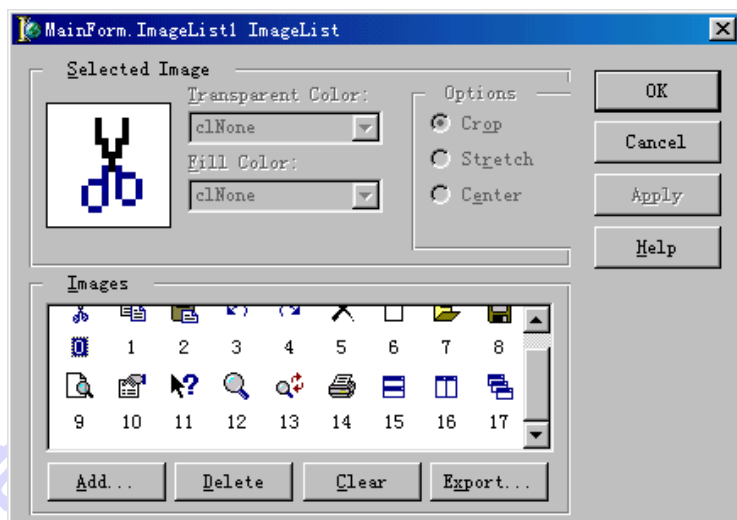


图 14-20 ImageList1 编辑器

在 Images 一栏的列表中，列出了该组件所有的图标及编号。

下面为没有图标的快捷按钮设置图标。选择右边第三个按钮，然后将其 ImageIndex 属性设置为 16，表示使用 ImageList1 中编号为 16 的图标；选择右边第二个按钮，然后将其 ImageIndex 属性设置为 15。

5. 现在运行程序并打开 3 个文档子窗口，此时主界面的显示如图 14-21 所示。

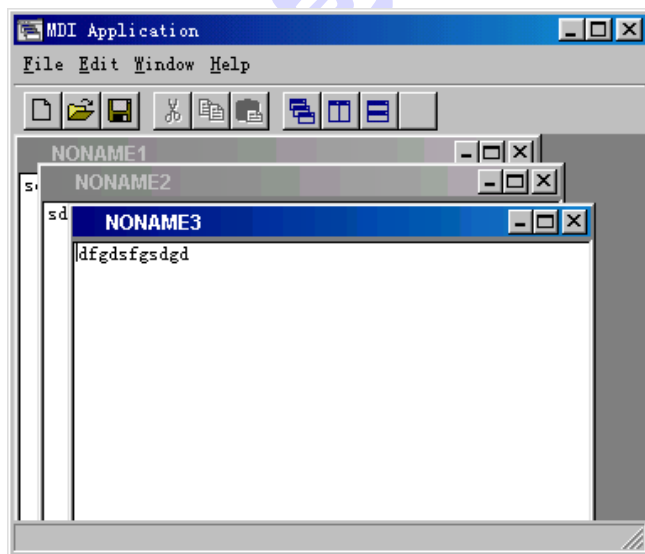


图 14-21 打开 3 个文档子窗口的主界面

6. 主表单对应的单元文件完整的源代码如下：

```
unit CLXMain;

interface

uses SysUtils, Classes, QForms, QImgList, QStdActns, QActnList, QDialogs,
```

```

    QMenus, QTypes, QComCtrls, QControls, QExtCtrls;

type
    TMainForm = class(TForm)
        MainMenu1: TMainMenu;
        File1: TMenuItem;
        FileNewItem: TMenuItem;
        FileOpenItem: TMenuItem;
        FileCloseItem: TMenuItem;
        Window1: TMenuItem;
        Help1: TMenuItem;
        N1: TMenuItem;
        FileExitItem: TMenuItem;
        WindowCascadeItem: TMenuItem;
        WindowTileItem: TMenuItem;
        HelpAboutItem: TMenuItem;
        OpenDialog: TOpenDialog;
        FileSaveItem: TMenuItem;
        FileSaveAsItem: TMenuItem;
        Edit1: TMenuItem;
        CutItem: TMenuItem;
        CopyItem: TMenuItem;
        PasteItem: TMenuItem;
        WindowMinimizeItem: TMenuItem;
        StatusBar: TStatusBar;
        ActionList1: TActionList;
        EditCut1: TEditCut;
        EditCopy1: TEditCopy;
        EditPaste1: TEditPaste;
        FileNew1: TAction;
        FileSave1: TAction;
        FileExit1: TAction;
        FileOpen1: TAction;
        FileSaveAs1: TAction;
        WindowCascadel: TWindowCascade;
        WindowMinimizeAll1: TWindowMinimizeAll;
        HelpAbout1: TAction;
        FileClose1: TWindowClose;
        WindowTileItem2: TMenuItem;
        ToolBar2: TToolBar;
        ToolButton1: TToolButton;
        ToolButton2: TToolButton;
        ToolButton3: TToolButton;
        ToolButton4: TToolButton;
        ToolButton5: TToolButton;
        ToolButton6: TToolButton;
        ToolButton9: TToolButton;
        ToolButton7: TToolButton;
        ToolButton8: TToolButton;
        ToolButton10: TToolButton;
    end;

```

```

    ToolButton11: TToolButton;
    WindowClose1: TWindowClose;
    WindowTile1: TWindowTile;
    ToolButton12: TToolButton;
    ImageList1: TImageList;
    SaveDialog1: TSaveDialog;
    procedure FileNew1Execute(Sender: TObject);
    procedure FileOpen1Execute(Sender: TObject);
    procedure HelpAbout1Execute(Sender: TObject);
    procedure FileExit1Execute(Sender: TObject);
    procedure FileSave1Execute(Sender: TObject);    //新添加的
    procedure FileSaveAs1Execute(Sender: TObject); //新添加的
private
    { Private declarations }
    FileName: string;                               //新添加的
    procedure CreateMDIChild(const Name: string);
public
    { Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.xfm}

uses clxchildwin, clxabout;

procedure TMainForm.CreateMDIChild(const Name: string);
var
    Child: TMDIChild;
begin
    { 建立一个新的MDI子窗口 }
    Child := TMDIChild.Create(Application);
    Child.Caption := Name;
    if FileExists(Name) then Child.Memo1.Lines.LoadFromFile(Name);
end;

procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
    CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
    FileName := TMDIChild(ActiveMDIChild).Caption;    //新添加的
end;

procedure TMainForm.FileOpen1Execute(Sender: TObject);
begin
    if OpenDialog.Execute then
        CreateMDIChild(OpenDialog.FileName);
end;

```



```

procedure TMainForm.HelpAbout1Execute(Sender: TObject);
begin
    AboutBox.ShowModal;
end;

procedure TMainForm.FileExit1Execute(Sender: TObject);
begin
    Close;
end;

procedure TMainForm.FileSave1Execute(Sender: TObject);           //新添加的
begin
    if (FileName = 'NONAME' + IntToStr(MDICHildCount)) then
        FileSaveAs1Execute(Sender)
    else
        TMDICHild(ActiveMDICHild).Memo1.Lines.SaveToFile(FileName);
end;

procedure TMainForm.FileSaveAs1Execute(Sender: TObject);         //新添加的
begin
    SaveDialog1.FileName := ActiveMDICHild.Caption;
    SaveDialog1.InitialDir := ExtractFilePath(FileName);
    if SaveDialog1.Execute then
    begin
        if ActiveMDICHild is TMDICHild then
        begin

TMDICHild(ActiveMDICHild).Memo1.Lines.SaveToFile(SaveDialog1.FileName);
            FileName := SaveDialog1.FileName;
            ActiveMDICHild.Caption := SaveDialog1.FileName;
            StatusBar.Panels[0].Text := FileName;
        end;
    end;
end;

end.

```

到此，我们的多文档应用程序设计完成。